SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Oracle User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Monk Version

SeeBeyond Proprietary and Confidential

The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20051014081020.

Contents

Chapter 1

Introduction 8

Using SQL Statements 9

Intended Reader 9

Components 9

Supported Operating Systems 9

System Requirements 10

External System Requirements 10

Chapter 2

Installation 11

Installation Overview 11

Installation Decisions 11

Installing Client and Network Components on Windows 12

Installing the Oracle e*Way on Windows 12

Pre-installation 12

Installation Procedure 13

Files/Directories Created by the Installation 14

Installing the Oracle e*Way on UNIX 14

Pre-installation 14

Installation Procedure 15

Oracle Network Components 16

SQL*Net Configuration files 17

Testing the SQL*Net Configuration 19

Troubleshooting Checklist 19

Setting up the Shared Library Search Path 20

Creating the Oracle e*Way Database User Account 21

Installing the Oracle e*Way with AIX 5.2 and AIX 5.3 21

Chapter 3

Configuration 22

Configuration Overview 22

e*Way Configuration Parameters 22
General Settings 23
Journal File Name 23
Max Resends Per Message 23
Max Failed Messages 24
Forward External Errors 24
Communication Setup 24
Start Exchange Data Schedule 24
Stop Exchange Data Schedule 25
Exchange Data Interval 25
Down Timeout 26
Up Timeout 26
Resend Timeout 26
Zero Wait Between Successful Exchanges 26
Monk Configuration 27
Basic e*Way Processes 28
How to Specify Function Names or File Names 34
Additional Path 35
Auxiliary Library Directories 35
Monk Environment Initialization File 35
Startup Function 36
Process Outgoing Message Function 37
Exchange Data with External Function 37
External Connection Establishment Function 38
External Connection Verification Function 39
External Connection Shutdown Function 39
Positive Acknowledgment Function 40
Negative Acknowledgment Function 40
Shutdown Command Notification Function 41
Database Setup 41
Database Type 41
Database Name 42
User Name 42
Encrypted Password 42

External Configuration Requirements 42
Configuring the Oracle Environment 42

Chapter 4

Implementation 44

Using the ETD Editor’s Build Tool 44
The Event Type Definition Files 47
Table or View 47
Dynamic SQL Statement 50
Stored Procedure 52

Sample One—Publishing e*Gate Events to an Oracle Database 53
Create the Schema 55
Create the Event Type Definitions 56
Add the Event Types 57
Create the Monk Scripts 58
Add and Configure the e*Ways 59
Add the IQs 61
Create the Collaboration Rules 61
Add and Configure the Collaborations 62
Run the Schema 63

Sample Two—Polling from an Oracle Database 64
Create the Schema 67
Create the Event Type Definitions 67
Add the Event Types 68
Create the Monk Scripts 69
Add and Configure the e*Ways 71
Add the IQs 73
Create the Collaboration Rules 74
Add and Configure the Collaborations 74
Run the Schema 76

Chapter 5

Oracle e*Way Functions 78

Basic Functions 78
event-send-to-egate 79
get-logical-name 80
send-external-down 81
send-external-up 82
shutdown-request 83
start-schedule 84
stop-schedule 85

Standard e*Way Functions 86
db-stdver-conn-estab 87
db-stdver-conn-shutdown 89
db-stdver-conn-ver 90
db-stdver-data-exchg 92
db-stdver-data-exchg-stub 93
db-stdver-init 94
db-stdver-neg-ack 96
db-stdver-pos-ack 97
db-stdver-proc-outgoing 98
db-stdver-proc-outgoing-stub 100
db-stdver-shutdown 102
db-stdver-startup 103

General Connection Functions 104
connection-handle? 105
db-alive 106
db-commit 108
db-get-error-str 109
db-login 111
db-logout 113
db-max-long-data-size 114
db-rollback 115

db-std-timestamp-format 116
make-connection-handle 117
statement-handle? 118

Static SQL Functions 119

Static vs. Dynamic SQL Functions 119
Oracle SQL Type Support 122
db-sql-column-names 123
db-sql-column-types 125
db-sql-column-values 127
db-sql-execute 129
db-sql-fetch 130
db-sql-fetch-cancel 131
db-sql-format 132
db-sql-select 134

Dynamic SQL Functions 135

db-stmt-bind 136
db-stmt-bind-binary 137
db-stmt-column-count 138
db-stmt-column-name 139
db-stmt-column-type 140
db-stmt-execute 141
db-stmt-fetch 142
db-stmt-fetch-cancel 143
db-stmt-param-assign 144
db-stmt-param-bind 145
db-stmt-param-count 146
db-stmt-param-type 147
db-stmt-row-count 148

Stored Procedure Functions 149

db-proc-bind 151
db-proc-bind-binary 152
db-proc-column-count 153
db-proc-column-name 155
db-proc-column-type 157
db-proc-execute 159
db-proc-fetch 161
db-proc-fetch-cancel 163
db-proc-max-records 164
db-proc-param-assign 165
db-proc-param-count 167
db-proc-param-io 169
db-proc-param-name 170
db-proc-param-type 171
db-proc-param-value 172
db-proc-return-exist 174
db-proc-return-type 176
db-proc-return-value 178

Message Event Functions 180

db-struct-bulk-insert 181
db-struct-call 182
db-struct-execute 183
db-struct-fetch 184
db-struct-insert 186
db-struct-select 188
db-struct-update 190

Sample Monk Scripts 192

Initializing Monk Extensions 193
Calling Stored Procedures 194
Inserting Records with Dynamic SQL Statements 196

Updating Records with Dynamic SQL Statements 198
Selecting Records with Dynamic SQL Statements 200
Deleting Records with Dynamic SQL Statements 202
Inserting a Binary Image to a Database 203
Retrieving an Image from a Database 206
Common Supporting Routines 208

Index 211

Chapter 1

Introduction

SeeBeyond™ developed the e*Way Intelligent Adapter for Oracle as a graphically-configurable e*Way. The Oracle e*Way implements the logic that sends Events (data) to e*Gate and queues the next Event for processing and transport to the database.

A Monk database access library is available to log into the database, issue Structured Query Language (SQL) statements, and call stored procedures. The Oracle e*Way uses Monk to execute user-supplied database access Monk scripts to retrieve information from or send information to a database. The fetched data (information) can be returned in a Monk Collaboration which simplifies the accessibility of each column in the database table. This document describes how to install and configure the Oracle e*Way.

Note: *For information on the Java-enabled e*Way Intelligent Adapter for Oracle, see the* ***e*Way Intelligent Adapter for Oracle User's Guide (Java-enabled)—Oracle_eWay_Java.pdf****.*

This Chapter Explains:

- **"Using SQL Statements" on page 9**
- **"Intended Reader" on page 9**
- **"Components" on page 9**
- **"Supported Operating Systems" on page 9**
- **"System Requirements" on page 10**

1.1 Using SQL Statements

The Oracle e*Way uses a Monk extension library to issue SQL statements. The library contains functions to access the database and generate SQL statements. SQL is the language used to communicate with the database server to access and manipulate data. By populating a database with the data flowing through an integration engine, all the information available to an integrated delivery network (IDN) is stored for evaluation. This allows the Oracle e*Way to operate independently of the underlying DBMS (database management system).

To access the database, you execute an SQL command, which is the American National Standards Institute (ANSI) standard language for operating upon relational databases. The language contains a large set of operators for defining and manipulating tables. SQL statements can be used to create, alter, and drop tables from a database.

1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have expert-level knowledge of Windows and/or UNIX operations and administration; to be thoroughly familiar with **Oracle** and to be thoroughly familiar with Windows-style GUI operations.

1.3 Components

The Oracle e*Way is comprised of the following:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk external function scripts
- e*Way Monk Functions

A complete list of installed files appears in **Table 1 on page 14** and **Table 2 on page 16**.

1.4 Supported Operating Systems

The Oracle e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP Tru64 V5.1A and 5.1B
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5L versions 5.1, 5.2, and 5.3

- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Red Hat Linux 8 (Intel x86)
- Sun Solaris 8 and 9
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Japanese IBM AIX 5L versions 5.1, 5.2, and 5.3
- Japanese Sun Solaris 8 and 9
- Korean Windows 2000, Windows XP, and Windows Server 2003
- Korean HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Korean IBM AIX 5L versions 5.1, and 5.2
- Korean Sun Solaris 8 and 9

1.5 System Requirements

To use the Oracle e*Way, you need the following:

- An e*Gate Participating Host.
- A TCP/IP network connection.

The client components of the databases with which the e*Way interfaces have their own requirements; see the appropriate client external documentation for more details.

1.5.1 External System Requirements

To enable the e*Way to communicate properly with the external system, the following are required:

- A database: Oracle 8i with patch 8.1.7.6 or 9i release 9.2.0.
- The Oracle client library must be installed on a Windows operating system to utilize the build tool.

Chapter 2

Installation

This chapter describes the procedures for installing the Oracle e*Way on both Windows and UNIX systems. A list of the files and directories created by the installation are included.

This Chapter Explains:

- **"Installation Overview" on page 11**
- **"Installing the Oracle e*Way on Windows" on page 12**
- **"Installing the Oracle e*Way on UNIX" on page 14**
- **"Oracle Network Components" on page 16**

Note: *Open and review the Readme.txt for any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM.*

2.1 Installation Overview

The installation procedure depends upon the operating system of the Participating Host on which you are installing this e*Way. You must have Administrator privileges to install this e*Way on either Windows or UNIX.

2.1.1 Installation Decisions

This section presents decisions to be made before beginning the installation. These decisions apply to both UNIX and Windows:

1. The operating system/platform on which the Oracle e*Way will operate.
2. The database network software required to operate the Oracle e*Way.
   - SQL*Net8

Note: *While the client version need not be the same as the server version it connects to, specific additional Oracle products may require a corresponding Net8 or SQL *Net release. Please refer to Oracle's Net8 Quick Reference Card for more information.*

3. The Oracle networking options to be installed.

   On UNIX:

- SQL*Net8
- TCP/IP Protocol Adaptor

On Windows:

- SQL*Net8
- TCP/IP Protocol Adapter
- OCI (Oracle Call Interface)

Issue the following command to determine which version of SQL*Net is installed:

- On UNIX:

```
echo $ORACLE_HOME
/opt/oracle/app/oracle/product/8.1.6
```

The output shows that SQL *Plus Version 8.1.6 is installed.

- On Windows:

From Program Files, go to the directory where Oracle for Windows is installed. If Plus80 appears, version 8.x.x is installed.

2.1.2 Installing Client and Network Components on Windows

The following Networking Options must be installed and configured before running the Oracle e*Way:

- The Oracle Client Oracle8i or 9i
- SQL*Net8 for Oracle8i
- TCP/IP Protocol Adapter
- OCI (Oracle Call Interface)

Note: *The Oracle e*Way requires a 32-bit version of the Oracle Client. The 64-bit Oracle Client is not compatible with this e*Way.*

2.2 Installing the Oracle e*Way on Windows

2.2.1 Pre-installation

1. Exit all Windows programs before running the setup program, including any anti-virus applications.
2. You must have Administrator privileges to install this e*Way.

2.2.2 Installation Procedure

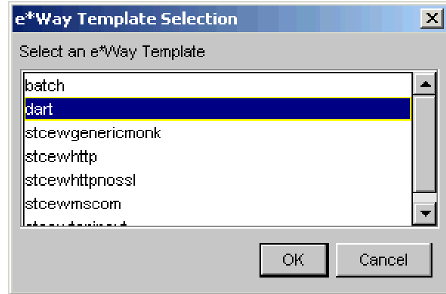
To install the Oracle e*Way on a Windows system

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
4. The InstallShield setup application will launch. Follow the on-screen instructions to install the e*Way.

Note: *Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

5. After the installation is complete, exit the install utility and launch the Schema Designer.
6. In the Component editor, create a new e*Way.
7. Display the new e*Way's properties.
8. On the General tab, under **Executable File**, click **Find**.
9. Select the file **stcewgenericmonk.exe**.
10. Under **Configuration file**, click **New**.
11. From the **Select an e*Way template** list, select **dart** and click **OK**.

Figure 1 e*Way Template Selection



12. The e*Way Editor will launch. Make any necessary changes, then save the configuration file.
13. You will return to the e*Way's property sheet. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 3**.

Note: *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the online Help system.*

*For more information about configuring e*Ways or how to use the e*Way Editor, see the **Working with e*Ways** chapter in the **e*Way Integrator User's Guide**.*

2.2.3 Files/Directories Created by the Installation

The Oracle e*Way CD-ROM contains the following files, which the Install Shield Wizard copies to the indicated directories on your computer, creating them if necessary.

Table 1 Installation Directories and Files (Windows)

Install Directory	Files
bin\	stcewgenericmonk.exe stcstruct.exe stc_dbapps.dll stc_dbmonkext.dll stc_dbora8.dll stc_dbora8i.dll stc_dbora9i.dll
configs\stcewgenericmonk\	dart.def dartRule.txt
monk_library	dart.gui
monk_library\dart\	db-struct-bulk-insert.monk db-struct-call.monk db-struct-execute.monk db-struct-fetch.monk db-struct-insert.monk db-struct-select.monk db-struct-update.monk db-stdver-eway-funcs.monk oramsg.ssc oramsg-display.monk db_bind.monk

2.3 Installing the Oracle e*Way on UNIX

2.3.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.

2.3.2 Installation Procedure

To install the Oracle e*Way on a UNIX system:

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type

   cd /cdrom

4. Start the installation script by typing:

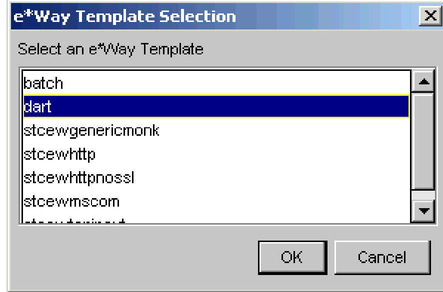
   setup.sh

5. A menu of options will appear. Select the "install e*Way" option. Then, follow any additional on-screen directions.

Note: *Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

6. After installation is complete, exit the installation utility and launch the Schema Designer.
7. In the Component editor, create a new e*Way.
8. Display the new e*Way's properties.
9. On the General tab, under **Executable File**, click **Find**.
10. Select the file **stcewgenericmonk.exe**.
11. Under **Configuration file**, click **New**.
12. From the **Select an e*Way template** list, select **dart** and click **OK**.

Figure 2 e*Way Template Selection



13. The e*Way Editor will launch. Make any necessary changes, then save the configuration file.
14. You will return to the e*Way's property sheet. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 3**.

Note: *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the online Help system.*

*For more information about configuring e*Ways or how to use the e*Way Editor, see the **Working with e*Ways** chapter in the **e*Way Integrator User's Guide**.*

The CD-ROM contains the following files, which are copied to the indicated path on your computer.

Table 2 Installation Directories and Files (UNIX)

Install Directory	Files
bin/	stcewgenericmonk.exe stcstruct.exe stc_dbapps.dll stc_dbmonkext.dll stc_dbora8.dll stc_dbora8i.dll stc_dbora9i.dll
lib/	stc_dbora8.dll stc_dbora9i.dll stc_monkfilesys.dll
configs/stcewgenericmonk/	dart.def
monk_library	dart.gui
monk_library/dart/	db-struct-bulk-insert.monk db-struct-call.monk db-struct-execute.monk db-struct-fetch.monk db-struct-insert.monk db-struct-select.monk db-struct-update.monk db-stdver-eway-funcs.monk oramsg.ssc oramsg-display.monk db_bind.monk

2.4 Oracle Network Components

The following Networking Options must be installed and configured before running the Oracle e*Way.

- SQL*Net8
- TCP/IP Protocol Adapter

- SQL*Plus (Recommended)

SQL*Plus is not required for the Oracle e*Way, but is helpful in testing connections and diagnosing connection problems.

2.4.1 SQL*Net Configuration files

Before you can configure SQL*Net8 you must have the following files ready:

- listener.ora
- tnsnames.ora
- sqlnet.ora

Example Listener configuration file—listener.ora

```
# LISTENER.ORA Configuration
File:/opt/oracle/app/oracle/product/8.1.6/network/admin/listener.ora
# Generated by Oracle configuration tools.

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = circe)(PORT = 1521))
      )
    )
    (DESCRIPTION =
      (PROTOCOL_STACK =
        (PRESENTATION = GIOP)
        (SESSION = RAW)
      )
      (ADDRESS = (PROTOCOL = TCP)(HOST = circe)(PORT = 2481))
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /opt/oracle/app/oracle/product/8.1.6)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (ORACLE_HOME = /opt/oracle/app/oracle/product/8.1.6)
      (SID_NAME = orcl816)
    )
  )
```

LISTENER is the default listener name, which is recommended by Oracle in a standard installation that requires only one listener on a machine.

The listener address section ADDRESS specifies what address to listen to. The listener listens for inter-process calls (IPCs) as well as calls from other nodes.

Two IPC addresses are created for each database that a listener listens to. In one, the key value is equal to the service name (for example, finance.world). It is used for

connections from other applications on the same node. The other IPC address (for example, orcl) is used by the database dispatcher to identify the listener.

For communication with other nodes, listener listens to the host (for example, finance.company.com) at a particular port (for example, 1521) using the specified protocol (for example, TCP/IP).

The section SID_LIST_LISTENER is used to list the SID (system identifier) of the databases (for example, orcl) on which the listener listens. The service name (for example, finance.world) is used as the global name.

The control parameter STARTUP_WAIT_TIME_LISTENER sets the number of seconds that the listener sleeps before responding to the first listener control status command. This feature assures that a listener with a slow protocol will have had time to start up before responding to a status request. The default is 0.

CONNECT_TIMEOUT_LISTENER sets the number of seconds that the listener waits to get a valid SQL*Net connection request before dropping the connection.

TRACE_LEVEL_LISTENER indicates the level of detail the trace facility records for listener events. ADMIN is the highest.

Example Client file—tnsnames.ora

```
# TNSNAMES.ORA Configuration
File:/opt/oracle/app/oracle/product/8.1.6/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

CIRCE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = circe)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = orcl816))
  )
ENIGMA =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = enigma)(PORT = 1521))
    (CONNECT_DATA = (SID = orcl8))
  )
LAMBDA =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(Host = lambda)(Port = 1521))
    (CONNECT_DATA = (SERVICE_NAME = LAMBDA))
  )
```

All connect distributors are assigned service names (for example, CIRCE). The user specifies the service name to identify the service to which the user wants to connect. The ADDRESS section specifies the listener address. See listener.ora above for listener address.

The CONNECT_DATA section specifies the SID (system identifier) or SERVICE_NAME by the remote database. This setting will vary depending upon the version of the Oracle server.

Example Network component file

The name of this file is sqlnet.ora

```
################
# Filename......: sqlnet.ora
# Node..........: local.world
# Date..........: 24-MAR-98 13:21:20
################
AUTOMATIC_IPC = OFF
TRACE_LEVEL_CLIENT = OFF
names.directory_path = (TNSNAMES)
names.default_domain = world
name.default_zone = world
sqlnet.expire_time = 10
```

The sqlnet.expire_time parameter determines how often SQL*Net sends a probe to verify that a client-server connection is still active. A value of 10 (minutes) is recommended by Oracle.

After you have generated the required configuration files, do the following:

- On the server side, move all three files to:

  $ORACLE_HOME/network/admin

- On the client side, distribute tnsnames.ora and sqlnet.ora and put them in:

  $ORACLE_HOME/network/admin

Verify that the file /etc/services has the entry LISTENER 1521/tcp.

2.4.2 Testing the SQL*Net Configuration

Before you can use SQL*Net with the server, you need to start a listener on the server. A listener is used by SQL*Net on the server side to receive an incoming connection from SQL*Net clients.

To start a listener, enter the following command on the server:

```
lsnrctl start
```

When you are running as a client, if the listener starts up successfully, you can use SQL*Plus on the client side to test whether SQL*Net is configured properly by establishing a connection with the server. The syntax of the command is:

```
sqlplus <user name>/<password>@<service name>
```

For example, if the Oracle server has user

```
sqlplus dart/dart@oracle.world
```

This command will start up SQL*Plus in the client machine and connect to the server specified by oracle.world as user <user name> with password <password>. The $ORACLE_HOME/network/admin/tnsnames.ora defines the service name for each Oracle data source.

2.4.3 Troubleshooting Checklist

- Ensure that you have protocol-level connectivity

Action: Use the PING utility to verify TCP/IP connectivity

- Ensure that the configuration files, client file and network files are in the proper directory.

  Actions:

  A Verify that tnsnames.ora exists on the client machine in the $ORACLE_HOME/network/ directory.

  B Verify that listener.ora exists on the server machine in the $ORACLE_HOME/network/ directory.

  C Verify that sqlnet.ora exists on the server machine in the $ORACLE_HOME/network/ directory.

- Ensure that the listener is "listening" for the same protocol that the client is trying to connect through.

  Action: Verify that listener.ora and tnsnames.ora specify the same protocol.

- Ensure that both server and client are running either Net8 or SQL *Net V2.

  Action: Run the Oracle Universal Installer to determine the version number.

- Ensure that you have the necessary Net8 protocol support installed.

  Action: Run the Oracle Universal Installer to determine that the correct product has been installed.

- Ensure that SQL*Net can recognize the host that must connect to if it is using TCP/IP.

  Action: If you are using TCP/IP, try replacing the HOST name in the net service name address with the IP address of the server machine.

For more information on specific error messages or technical bulletins on errors received when performing these diagnostics tests, refer:

- The Net8 Administrator's Guide

2.4.4 Setting up the Shared Library Search Path

The following sections provide detailed descriptions for setting the shared library search path used by both the Oracle e*way and the Oracle Open Client. The shared library search path follows:

HP-UX

```
SHLIB_PATH
```

AIX

```
LIBPATH
```

DEC

```
LD_LIBRARY_PATH
```

Solaris

```
LD_LIBRARY_PATH
```

2.4.5 Creating the Oracle e*Way Database User Account

Check to see if the Oracle server is running. If not, start the server and make sure the "listener" is running, do the following to create the Oracle e*Way database user in the Oracle database.

At the prompt type the following:

```
svrmgrl
connect internal
create user_name identified by password
```

After creating the user, privileges must be granted. Type the following:

```
grant dba to user name
```

2.5 Installing the Oracle e*Way with AIX 5.2 and AIX 5.3

If you are running Oracle 9i on and AIX 5.2 or AIX 5.3 operating system, you will need to do the following:

1. During the Registry installation process, select both the Windows operating system and the AIX operating system.
2. Manually create the directory monk_scripts\common on the Windows machine. This directory is necessary for the builder to save the .ssc file to.
3. On the AIX machine, edit the LIBPATH to include $ORACLE_HOME/lib32 from the default of $ORACLE_HOME/lib. This will change the default from 64 bits to 32 bits.

Chapter 3

Configuration

This chapter describes how to configure the Oracle e*Way by setting the configuration parameters using the e*Way Editor.

This Chapter Explains:

- **"Configuration Overview" on page 22**
- **"General Settings" on page 23**
- **"Communication Setup" on page 24**
- **"Monk Configuration" on page 27**
- **"Database Setup" on page 41**
- **"Configuring the Oracle Environment" on page 42**

3.1 Configuration Overview

Before you can run the Oracle e*Way, you must configure it using the e*Way Edit Settings window, which is accessed from the e*Gate Schema Designer GUI. The Oracle e*Way package includes a default configuration file which you can modify using this window.

3.2 e*Way Configuration Parameters

The e*Way configuration parameters are set using the e*Way Editor.

To change e*Way configuration parameters:

1. In the Schema Designer's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.

Note: *When creating a new e*Way, you must also select the* ***dart*** *template file from the* ***e*Way Template Selection*** *list.*

3 In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *Working with e*Ways* chapter in the *e*Way Integrator User's Guide*.

The e*Way's configuration parameters are organized into the following four sections:

- General Settings
- Communication Setup
- Monk Configuration
- Database Setup

3.2.1 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is stored in the e*Gate "SystemData" directory. See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

Additional Information

An Event is journaled for the following conditions:

- When the number of resends is exceeded (see **Max Resends Per Message** below).
- When its receipt is due to an external error, but **Forward External Errors** is set to **No**. (See **"Forward External Errors" on page 24** for more information.)

Max Resends Per Message

Description

Specifies the number of times the e*Way attempts to resend a message (Event) to the external system after receiving an error. When this maximum is reached, the message is considered "Failed" and is written to the journal file.

Required Values

An integer between 1 and 1,024. The default is 5.

Max Failed Messages

Description

Specifies the maximum number of failed messages (Events) that the e*Way allows. When the specified number of failed messages is reached, the e*Way shut downs and exits.

Required Values

An integer between 1 and 1,024. The default is 3.

Forward External Errors

Description

Selects whether error messages that begin with the string "DATAERR" that are received from the external system are queued to the e*Way's configured queue. If this parameter is set to **No**, then error messages will be ignored. See **"Exchange Data with External Function" on page 37** for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages is not forwarded. See **Figure 8 on page 32** for more information about how the e*Way uses this function.

3.2.2 Communication Setup

The Communication Setup parameters control the schedule by which the e*Way obtains data from the external system.

Note: *The schedule you set using the e*Way's properties (**Start Up** tab) in the Schema Designer controls when the e*Way executable runs. The schedule you set within the parameters discussed in this section (using the e*Way Editor) determines when data is exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Start Exchange Data Schedule

Description

Establishes the schedule to invoke the e*Way's **Exchange Data with External Function**.

Required Values

One of the following:

- One or more specific dates/times.
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Also required: If you set a schedule using this parameter, you must also define all three of the following:

- **Exchange Data With External Function**

- **Positive Acknowledgment Function**
- **Negative Acknowledgment Function**

If you do not do so, the e*Way terminates execution when the schedule attempts to start.

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an ACK or NAK to the external system (using the **Positive Acknowledgement Function** and **Negative Acknowledgement Function**) and whether the connection to the external system is active. If no ACK/NAK is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External Function**. Thereafter, the **Exchange Data with External Function** is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

See **"Exchange Data with External Function" on page 37**, **"Exchange Data Interval" on page 25**, and **"Stop Exchange Data Schedule" on page 25** for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

- One or more specific dates/times.
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Exchange Data Interval

Description

Specifies the number of seconds the e*Way waits between calls to the **Exchange Data with External Function** during scheduled data exchanges.

Required Values

An integer between 0 and 86,400. The default is 120.

Additional Information

If **Zero Wait Between Successful Exchanges** is set to **Yes** and the **Exchange Data with External Function** returns data, the **Exchange Data Interval** setting is ignored and the e*Way invokes the **Exchange Data with External Function** immediately.

If this parameter is set to zero, there is no exchange data schedule set and the **Exchange Data with External Function** is never called.

See **"Down Timeout" on page 26** and **"Stop Exchange Data Schedule" on page 25** for more information about the data-exchange schedule.

Down Timeout

Description

Specifies the number of seconds that the e*Way waits between calls to the **External Connection Establishment Function**. See **"External Connection Establishment Function" on page 38** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Up Timeout

Description

Specifies the number of seconds the e*Way waits between calls to the **External Connection Verification Function**. See **"External Connection Verification Function" on page 39** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Resend Timeout

Description

Specifies the number of seconds the e*Way waits between attempts to resend a message (Event) to the external system, after receiving an error message from the external system.

Required Values

An integer between 1 and 86,400. The default is 10.

Zero Wait Between Successful Exchanges

Description

Selects whether to initiate data exchange after the **Exchange Data Interval** or immediately after a successful previous exchange.

Required Values

Yes or **No**. The default is **No**.

Additional Information

If this parameter is set to **Yes** and the previous exchange function returned data, then the e*Way immediately invokes the **Exchange Data With External Function**. If this parameter is set to **No**, the e*Way always waits the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External Function**.

See **"Exchange Data with External Function" on page 37** for more information.

3.2.3 Monk Configuration

The parameters in this section help you set up the information required by the e*Way to utilize Monk for communication with the external system.

Architecturally, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers that handle communication with the external application, built upon an e*Way Kernel layer that manages the processing of data and subscribing or publishing to other e*Gate components (see Figure 3).

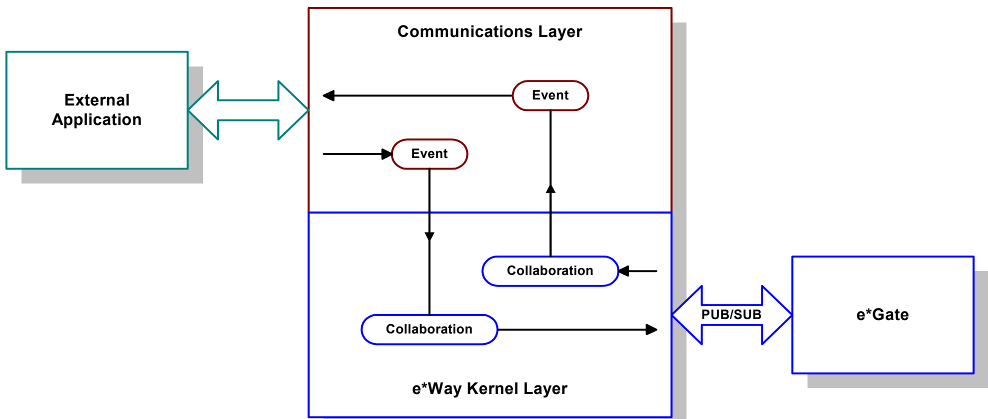
Figure 3 Typical e*Way Architecture

External Application
RFC
Typical e*Way
Additional Layer such as API Model
Communications Layer such as Remote Function Call (RFC) Transport
e*Way Kernel Layer
PUB/SUB
e*Gate

Each layer contains Monk scripts and/or functions, and makes use of lower-level Monk functions residing in the layer beneath. You, as user, primarily use the highest-level functions, which reside in the upper layer(s).

The upper layers of the e*Way use Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate “Events,” send those Events to Collaborations, and manage the connection between the e*Way and the external system (see Figure 4).

Figure 4 Basic e*Way Operations



Configuration options that control the Monk environment and define the Monk functions used to perform these basic e*Way operations are discussed in **Chapter 4**. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft Wordpad* or *Notepad*).

The upper layers of the e*Way are single-threaded. Functions run serially, and only one function can be executed at a time. The e*Way Kernel is multi-threaded, with one executable thread for each Collaboration. Each thread maintains its own Monk environment; therefore, information such as variables, functions, path information, and so on cannot be shared between threads.

The basic set of e*Way Kernel Monk functions is described in **Chapter 5**. Generally, e*Way Kernel Monk functions should be called directly only when there is a specific need not addressed by higher-level Monk functions, and should be used only by experienced developers.

Basic e*Way Processes

The Monk functions in the "communications half" of the e*Way fall into the following groups:

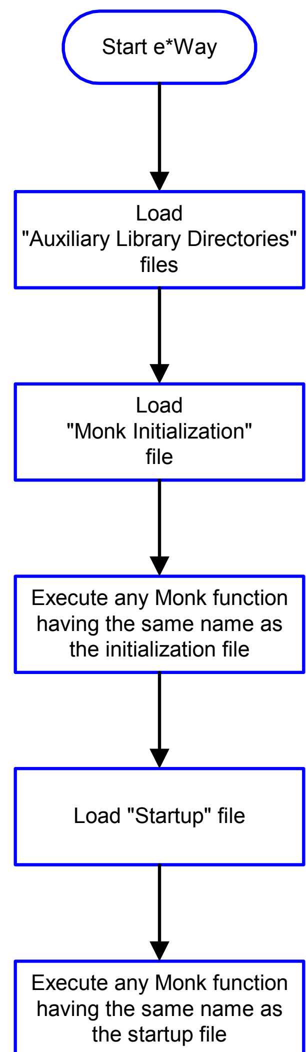
Type of Operation	Name
Initialization	**"Startup Function" on page 36** (also see **"Monk Environment Initialization File" on page 35**)
Connection	**"External Connection Establishment Function" on page 38** **"External Connection Verification Function" on page 39** **"External Connection Shutdown Function" on page 39**
Schedule-driven data exchange	**"Exchange Data with External Function" on page 37** **"Positive Acknowledgment Function" on page 40** **"Negative Acknowledgment Function" on page 40**
Shutdown	**"Shutdown Command Notification Function" on page 41**
Event-driven data exchange	**"Process Outgoing Message Function" on page 37**

A series of figures on the next several pages illustrate the interaction and operation of these functions.

Initialization Functions

Figure 5 illustrates how the e*Way executes its initialization functions.

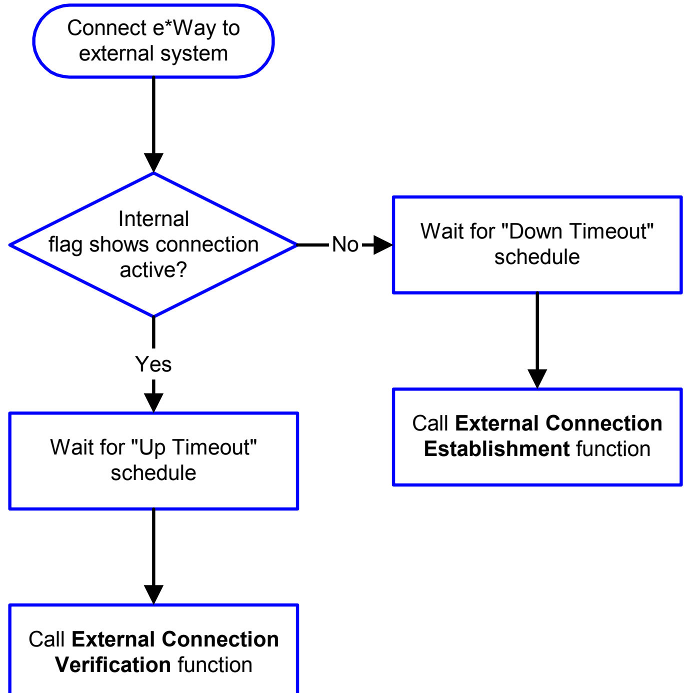
Figure 5 Initialization Functions



Connection Functions

Figure 6 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 6 Connection Establishment and Verification Functions



> ***Note:*** *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 8 on page 32** *and* **Figure 10 on page 34** *for examples of how different functions use this flag.*
>
> *User functions can manually set this flag using Monk functions. See* **send-external-up** *on page 82 and* **send-external-down** *on page 81 for more information.*

Figure 7 illustrates how the e*Way executes its "connection shutdown" function.

Figure 7 Connection Shutdown Function

Control Broker issues "Suspend" command

Call **External Connection Shutdown** function with parameter "SUSPEND_NOTIFICATION"

Return any value

e*Way closes connection

Schedule-driven Data Exchange Functions

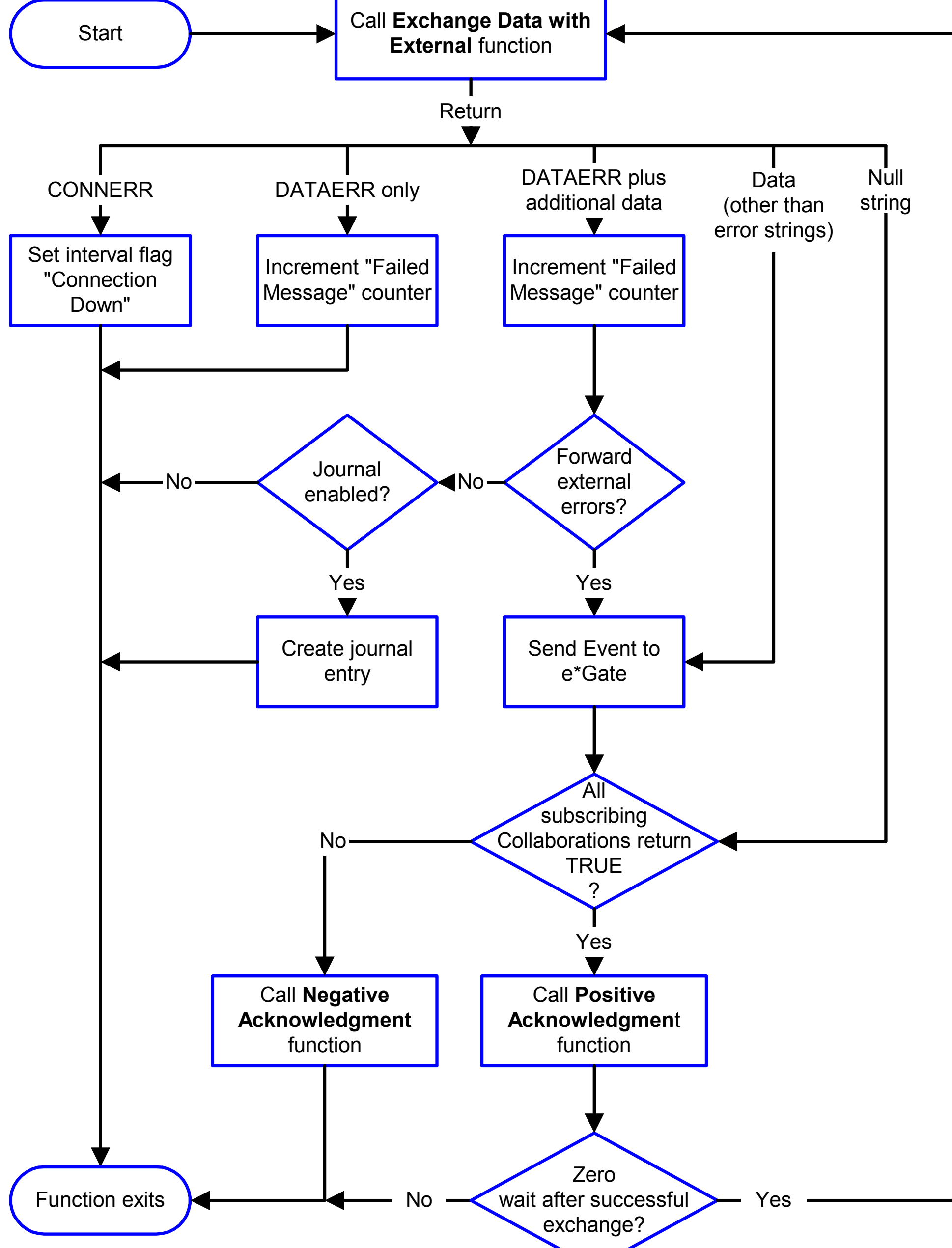
Figure 8 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

“Start” can occur in any of the following ways:

- The “Start Data Exchange” time occurs.
- Periodically during data-exchange schedule (after “Start Data Exchange” time, but before “Stop Data Exchange” time), as set by the Exchange Data Interval.
- The **start-schedule** Monk function is called.

After the function exits, the e*Way waits for the next “start schedule” time or command.

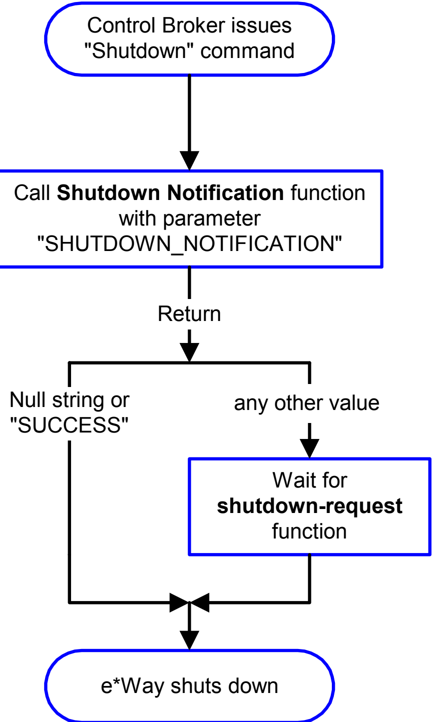
Figure 8 Schedule-Driven Data Exchange Functions



Shutdown Functions

Figure 9 illustrates how the e*Way implements the shutdown request function.

Figure 9 Shutdown Functions



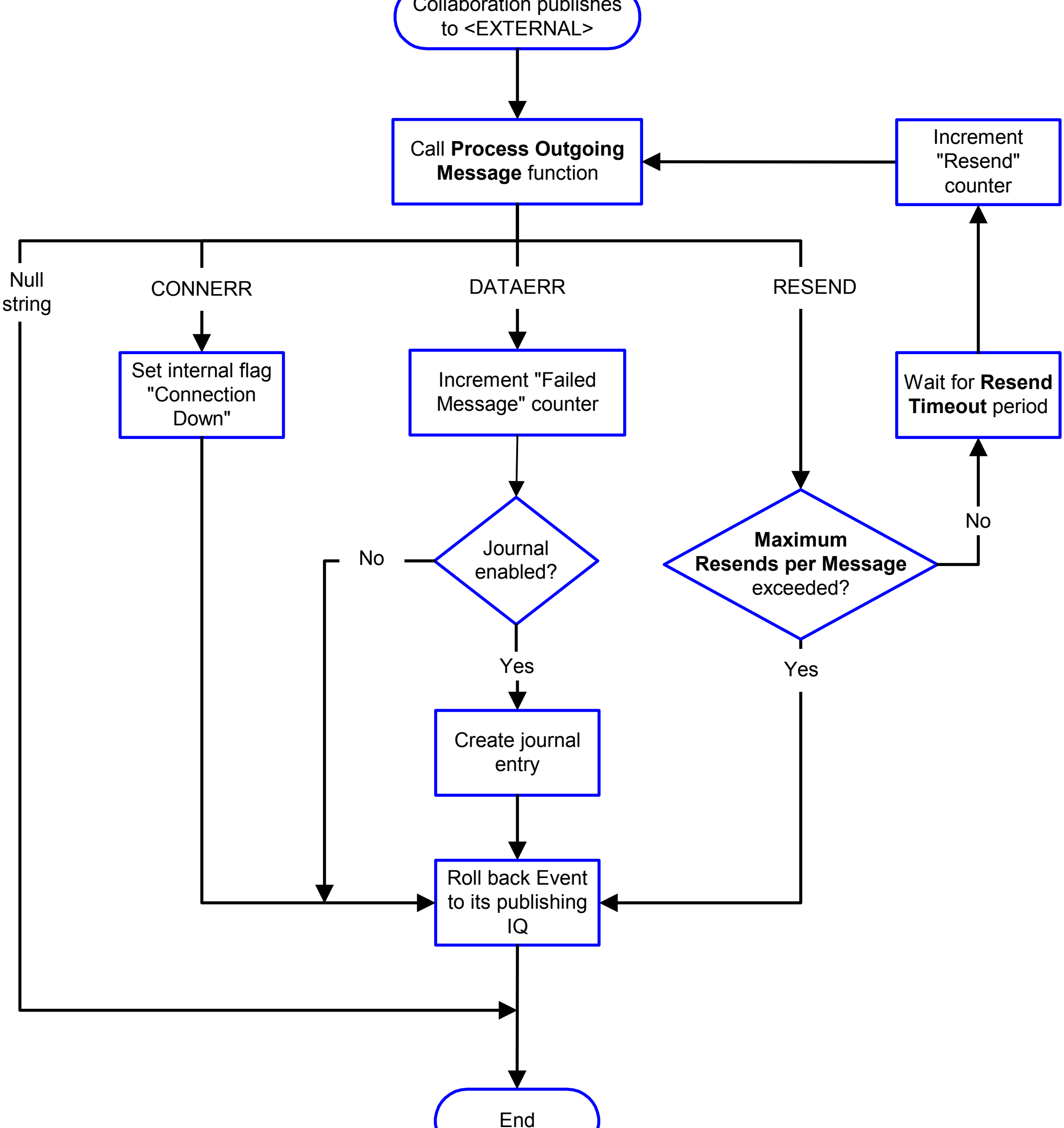
Event-driven Data Exchange Functions

Figure 10 on the next page illustrates event-driven data-exchange using the **Process Outgoing Message Function**.

Every two minutes, the e*Way checks the “Failed Message” counter against the value specified by the **Max Failed Messages** parameter. When the “Failed Message” counter exceeds the specified maximum value, the e*Way logs an error and shuts down.

After the function exits, the e*Way waits for the next outgoing Event.

Figure 10 Event-Driven Data-exchange Functions



How to Specify Function Names or File Names

Parameters that require the name of a Monk function accepts either a function name or a file name. If you specify a file name, be sure that the file has one of the following extensions:

- .monk
- .tsc
- .dsc

Additional Path

Description

Specifies a path to be appended to the "load path," the path Monk uses to locate files and data (set internally within Monk). The directory specified in **Additional Path** is searched after the default load path.

Required Values

A path name, or a series of paths separated by semicolons. This parameter is optional and may be left blank.

Additional information

The default load paths are determined by the "bin" and "Shared Data" settings in the .egate.store file. See the *e*Gate Integrator System Administration and Operations Guide* for more information about this file.

To specify multiple directories, manually enter the directory names rather than selecting them with the "file selection" button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths. For example:

```
monk_scripts\my_dir;c:\my_directory
```

The internal e*Way function that loads this path information is called only once, when the e*Way first starts up.

Auxiliary Library Directories

Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories are automatically loaded into the e*Way's Monk environment. This parameter is optional and may be left blank.

Required Values

A path name, or a series of paths separated by semicolons. The default is **monk_library/dart**.

Additional information

To specify multiple directories, manually enter the directory names rather than selecting them with the "file selection" button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths. For example:

```
monk_scripts\my_dir;c:\my_directory
```

The internal e*Way function that loads this path information is called only once, when the e*Way first starts up.

This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which are loaded after the **Auxiliary Library Directories** are loaded. Use this feature to initialize the

e*Way's Monk environment (for example, to define Monk variables that are used by the e*Way's function scripts).

Required Values

A filename within the "load path", or filename plus path information (relative or absolute). If path information is specified, that path is appended to the "load path." See **"Additional Path" on page 35** for more information about the "load path."

The default is **db-stdver-init**. See **db-stdver-init** on page 94 for more information.

Additional information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way loads this file and tries to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way would attempt to execute the function **my-init**).

Typically, it is a good practice to initialize any global Monk variables that may be used by any other Monk Extension scripts.

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 5 on page 29**).

Startup Function

Description

Specifies a Monk function that the e*Way loads and invokes upon startup or whenever the e*Way's configuration is reloaded. This function should be used to initialize the external system before data exchange starts.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is optional and may be left blank.

The default is **db-stdver-startup**. See **db-stdver-startup** on page 103 for more information.

Additional information

The function accepts no input, and must return a string.

The string "FAILURE" indicates that the function failed; any other string (including a null string) indicates success.

This function is called after the e*Way loads the specified **Monk Environment Initialization File** and any files within the specified **Auxiliary Library Directories**.

The e*Way loads this file and tries to invoke a function of the same base name as the file name (see **Figure 5 on page 29**). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

Process Outgoing Message Function

Description

Specifies the Monk function responsible for sending outgoing messages (Events) from the e*Way to the external system. This function is event-driven (unlike the **Exchange Data with External Function**, which is schedule-driven).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***You may not leave this field blank.***

The default is **db-stdver-proc-outgoing**. See **db-stdver-proc-outgoing** on page 98 for more information.

Additional Information

The function requires a non-null string as input (the outgoing Event to be sent) and must return a string.

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Schema Designer). The function returns one of the following (see **Figure 8 on page 32** for more details):

- Null string: Indicates that the Event was published successfully to the external system.
- "RESEND": Indicates that the Event should be resent.
- "CONNERR": Indicates that there is a problem communicating with the external system.
- "DATAERR": Indicates that there is a problem with the message (Event) data itself.

Note: *If you wish to use* ***event-send-to-egate*** *to enqueue failed Events in a separate IQ, the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See* **"event-send-to-egate" on page 79** *for more information.*

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is optional and may be left blank. However, this parameter is required if a schedule was set using the **Start Exchange Data Schedule** parameter. If so, you must also define the following:

- **Positive Acknowledgement Function**

- **Negative Acknowledgement Function**

The default is **db-stdver-data-exchg**. See **db-stdver-data-exchg** on page 92 for more information.

Additional Information

The function accepts no input and must return a string (see **Figure 10 on page 34** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information is sent into the e*Gate system.
- "CONNERR": Indicates that a problem with the connection to the external system has occurred.
- "DATAERR": Indicates that a problem with the data itself has occurred.
- Any other string: The contents of the string are packaged as an inbound Event. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

This function is initially triggered by the **Start Exchange Data Schedule** or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been ACKed or NAKed (by the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter. If this parameter is set to **Yes**, the e*Way immediately calls the **Exchange Data with External Function** again; otherwise, the e*Way does not call the function until the next scheduled "start exchange" time or the schedule is manually invoked using the Monk function **start-schedule** (see **start-schedule** on page 84 for more information).

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way calls when it has determined that the connection to the external system is down (or is unknown).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***This field cannot be left blank.***

The default is **db-stdver-conn-estab**. See **db-stdver-conn-estab** on page 87 for more information.

Additional Information

The function accepts no input and must return a string:

- "SUCCESS" or "UP": Indicates that the connection was established successfully.
- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Down Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Verification Function** (see below) is called when the e*Way has determined that its connection to the external system is up.

External Connection Verification Function

Description

Specifies a Monk function that the e*Way calls when its internal variables show that the connection to the external system is up.

Required Values

The name of a Monk function. This function is optional; if no **External Connection Verification Function** is specified, the e*Way executes the **External Connection Establishment Function** in its place.

The default is **db-stdver-conn-ver**. See **db-stdver-conn-ver** on page 90 for more information.

Additional Information

The function accepts no input and must return a string:

- "SUCCESS" or "UP": Indicates that the connection was established successfully.
- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Up Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Establishment Function** is called when the e*Way has determined that its connection to the external system is down or is unknown.

External Connection Shutdown Function

Description

Specifies a Monk function that the e*Way calls to shut down the connection to the external system.

Required Values

The name of a Monk function. This parameter is optional.

The default is **db-stdver-conn-shutdown**. See **db-stdver-conn-shutdown** on page 89 for more information.

Additional Information

This function requires a string as input, and may return a string.

This function is only invoked when the e*Way receives a "suspend" command from a Control Broker. When the "suspend" command is received, the e*Way invokes this function, and passes the string "SUSPEND_NOTIFICATION" as an argument.

Any return value indicates that the "suspend" command can proceed and that the connection to the external system can be broken immediately.

Positive Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls when *all* the Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External Function** is defined.

The default is **db-stdver-pos-ack**. See **db-stdver-pos-ack** on page 97 for more information.

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the **Positive Acknowledgment Function** is called again, with the same input data.
- Null string: The function completed execution successfully.

After the **Exchange Data with External Function** returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the **Positive Acknowledgment Function** (otherwise, the e*Way executes the **Negative Acknowledgment Function**).

Negative Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls when the e*Way fails to process and queue Events from the external system.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External Function** is defined.

The default is **db-stdver-neg-ack**. See **db-stdver-neg-ack** on page 96 for more information.

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.
- Null string: The function completed execution successfully.

This function is only called during the processing of inbound Events. After the **Exchange Data with External Function** returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is not completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the **Negative Acknowledgment Function** (otherwise, the e*Way executes the **Positive Acknowledgment Function**).

Shutdown Command Notification Function

Description

Specifies a Monk function that is called when the e*Way receives a "shut down" command from the Control Broker. This parameter is optional.

Required Values

The name of a Monk function.

The default is **db-stdver-shutdown**. See **db-stdver-shutdown** on page 102 for more information.

Additional Information

When the Control Broker issues a shutdown command to the e*Way, the e*Way calls this function with the string "SHUTDOWN_NOTIFICATION" passed as a parameter.

The function accepts a string as input and must return a string:

- A null string or "SUCCESS": Indicates that the shutdown can occur immediately.
- Any other string: Indicates that shutdown must be postponed. Once postponed, shutdown does not proceed until the Monk function **shutdown-request** is executed (see **shutdown-request** on page 83).

Note: *If you postpone a shutdown using this function, be sure to use the* (**shutdown-request)** *function to complete the process in a timely manner.*

3.2.4 Database Setup

Database Type

Description

Specifies the type of database.

Required Values

DB2, **ODBC**, **ORACLE8i**, **ORACLE9i**, **SYBASE11**, or **SYBASE12**

The default is **SYBASE11**. Change this value according to the Oracle client version used by your Oracle implementation.

Database Name

Description

The name of the database. Refer to the TNS service name as configured in the **tnsnames.ora** file.

Required Values

None. Any valid string.

User Name

Description

The name used to access the database.

Required Values

None. Any valid string.

Encrypted Password

Description

The password that provides access to the database.

Required Values

Any valid string.

3.3 External Configuration Requirements

This section describes environment variable requirements to support the Oracle e*Way.

3.3.1 Configuring the Oracle Environment

Make sure your database server has been set up and the following environment variables have been defined:

- ORACLE_HOME - This specifies where you installed Oracle.

  i.e., D:Oracle\Ora81

  This specifies where the Oracle Client is installed.
- ORACLE_SID - This specifies the name of the server.
- LD_LIBRARY_PATH - This specifies the library path and needs to include the path to the Oracle library, i.e., $ORACLE_HOME/lib. However, if you are using an HP-UX or AIX operating system with Oracle 9i, you will need to include the value "$ORACLE_HOME/lib32" (in stead of "$ORACLE_HOME/lib") in your library path environment variable.

Note: *You can define these environment variables in .cshrc in the C shell or .profile in the Korn/Bash shell.*

Chapter 4

Implementation

This chapter contains information explaining the use of the ETD Editor's Build Tool as well as two sample Oracle e*Way scenarios.

This Chapter Includes:

- **"Using the ETD Editor's Build Tool" on page 44**
- **"Sample One—Publishing e*Gate Events to an Oracle Database" on page 53**
- **"Sample Two—Polling from an Oracle Database" on page 64**

4.1 Using the ETD Editor's Build Tool

The Event Type Definition Editor's Build tool automatically creates an Event Type Definition file based on the tables in an existing database. The Event Type Definition (ETD) can be created based on one of (or a combination of) the following criteria:

- **Table or View** – Displays all of the columns in the specified table or view.
- **Dynamic SQL Statement** – Displays the format of the results of an SQL statement. This can be used to return only a few of the columns in a table.
- **Stored Procedure** – Displays the format of the results of an SQL Stored Procedure. This option is only available for *Delimited* messages.

The results of these three types of message criteria are explained in **"The Event Type Definition Files" on page 47**.

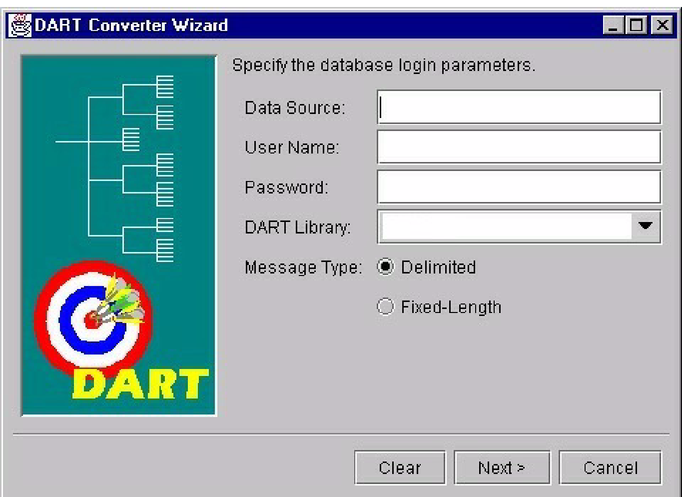
To create an Event Type Definition using the Build tool:

1. Launch the ETD (Event Type Definition) Editor.
2. On the ETD Editor's Toolbar, click **Build**.

   The **Build an Event Type Definition** dialog box appears.
3. In the File name box, type the name of the ETD file you wish to build. *Do not specify any file extension*—the Editor will supply an "ssc" extension for you.
4. Under **Build From**, select **Library Converter**.
5. Under **Select a Library Converter**, select DART Converter.
6. Click **OK**.

7 The Converter Wizard will launch.

Figure 11 Converter Wizard Subordinate Dialog Box



8 Enter the Data Source.

9 Enter the User Name.

10 Enter the Password.

11 Select the DART Library. You must have installed the corresponding e*Way prior to making your selection.

12 Select the correct Message Type.

Note: *It is important to enter the correct Data Source and Message Type. For Oracle the Data Source is in the Servicename.world format*
The Fixed-length Message Type is used for DART bulk insert only.
The Delimited Message Type is for all other DART structure calls.
See **Figure 11 on page 45**

If you select Delimited Message Type the following dialog box will appear.

Figure 12 Converter Wizard Delimited Message Type Dialog Box

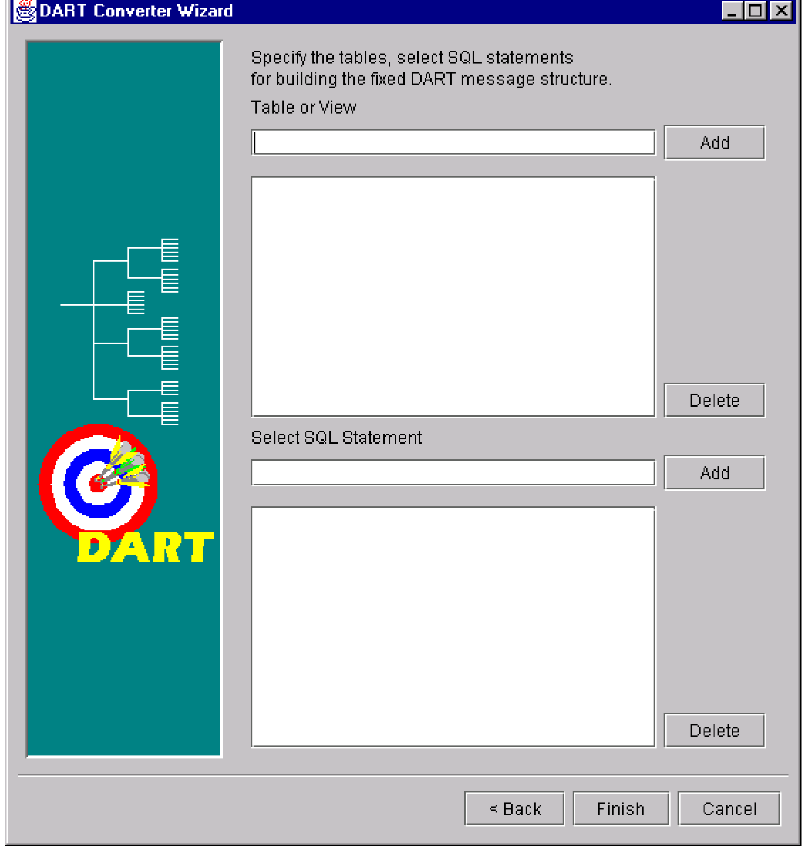
13 Select or Add the correct Table or View

14 Select or Add the correct SQL Statement

15 Select or Add the correct Stored Procedure.

If you select the Fixed-Length Message Type the following dialog box will appear.

Figure 13 Converter Wizard Fixed-Length Message Type Dialog Box



16 Select or Add the correct Table or View

17 Select or Add the correct SQL Statement

18 Edit or Finish your selections.

Note: *The (#) character cannot be used in the node name of the .ssc file. The Oracle e*Way will be unable to generate the correct node name for the column name of a table that contains the (#) character, as Monk will filter out the character.*

For Oracle, ($), or (#) can be used in a name, although the Oracle User's Guide strongly discourages their use.

4.1.1 The Event Type Definition Files

The DART Converter Build Tool will create a different ETD based on the criteria that was specified in the Build Tool Wizard (see **Figure 12 on page 46** and **Figure 13 on page 47**).

Table or View

Entering a table or view name as a selection criteria will display all of the columns in that table or view. This is useful when you want to access an entire record from the table

as an e*Gate Event. The criteria shown in Figure 14 generates the ETD shown in Figure 15.

Figure 14 Table or View Selection

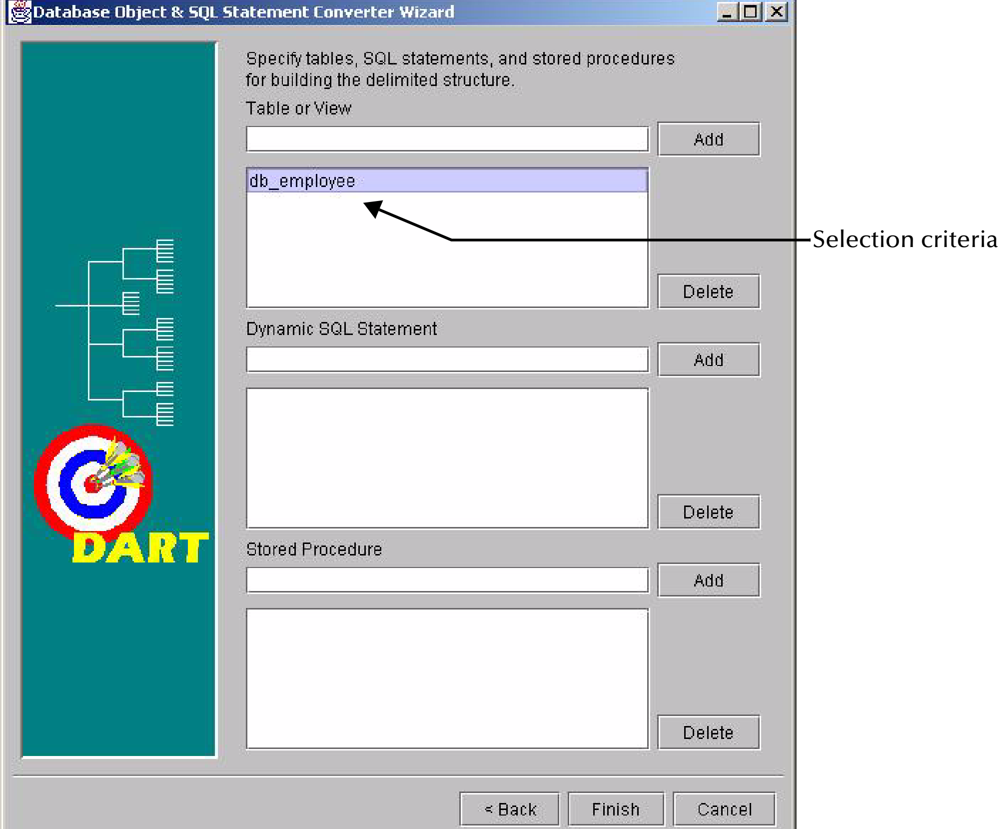


Figure 15 Table or View ETD

The ETD that is generated by the DART Converter Build Tool using the Table or View criteria contains the elements shown in the table below.

Table 3 Elements of the Table or View ETD

Element	Description
ETD Name	This is the root node of the Event Type Definition.
Table Name	This node displays the name of the table or view.
Column Name	This is the name of the column(s) in the selected table or view.
Field Value	This is the value of the data in the column. This can be thought of as the *payload data* for this column.
Data Type	This node designates the type of data contained in the value field.
Constraint Code	The constraint codes are based on the column constraints in the table. The possible codes are: ▪ **I** – *Insert* operations are allowed in this column. ▪ **U** – *Update* operations are allowed in this column. ▪ **N** – *Neither* insert nor update operations are allowed in this column. ▪ **B** – *Both* insert and update operations are allowed in this column.

Dynamic SQL Statement

Entering an SQL statement as a selection criteria will display the format of the results of that SQL statement. This is useful when you only want to access certain columns from the table for a particular e*Gate Event.

To use this type of ETD, you should use the **db-stmt-bind** function to bind the dynamic statement and **db-struct-execute** function to execute the SQL statement. For more information, see **db-stmt-bind** on page 136 and **db-struct-execute** on page 183.

The SQL statement shown in Figure 16 generates an ETD that returns specific records from the table based on the selection criteria (which is represented by a question mark "**?**"). The resulting ETD is shown in **Figure 17 on page 51**.

Note: *It is not necessary to include the terminating semi-colon as part of the SQL statement.*

Figure 16 Dynamic SQL Statement Selection

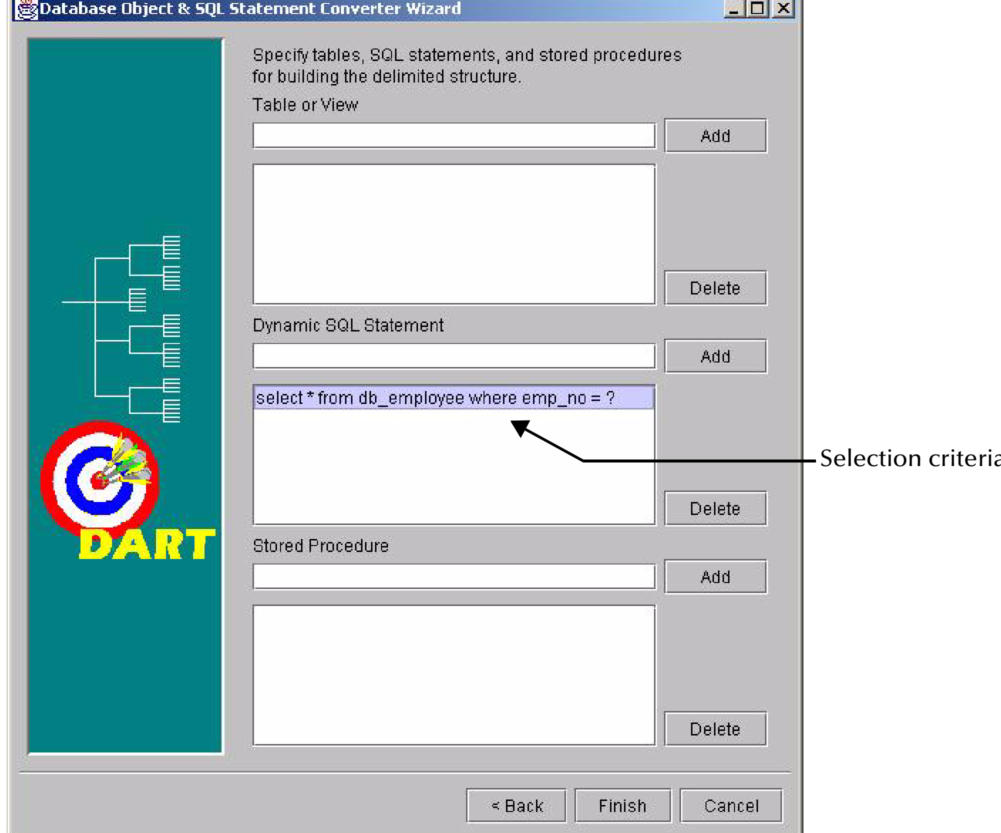
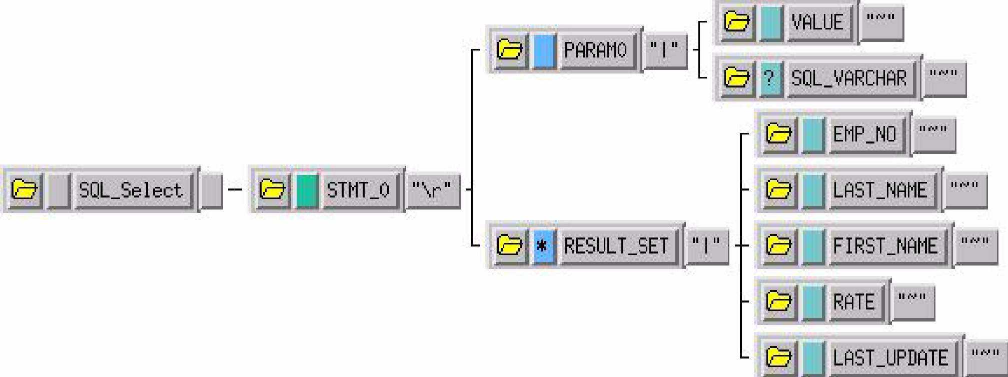


Figure 17 Dynamic SQL Statement ETD



The **PARAM0** node in the ETD shown in Figure 17 represents the criteria specified in the SQL statement. Additional criteria would be represented in additional nodes (**PARAM1**, **PARAM2**, and so forth). For example, using the following SQL statement:

```
SELECT * FROM db_employee WHERE last_name = ? AND first_name = ?
```

the Build Tool would generate an ETD with two input parameter nodes (**PARAM0** and **PARAM1**)—one for each of the criteria (**?**). The **VALUE** nodes of these input parameter nodes are used to carry the payload of the selection statement.

Stored Procedure

Entering a stored procedure name as a selection criteria will generate an ETD that will access a stored procedure in the external database. This is useful when you want to access the results of a stored procedure.

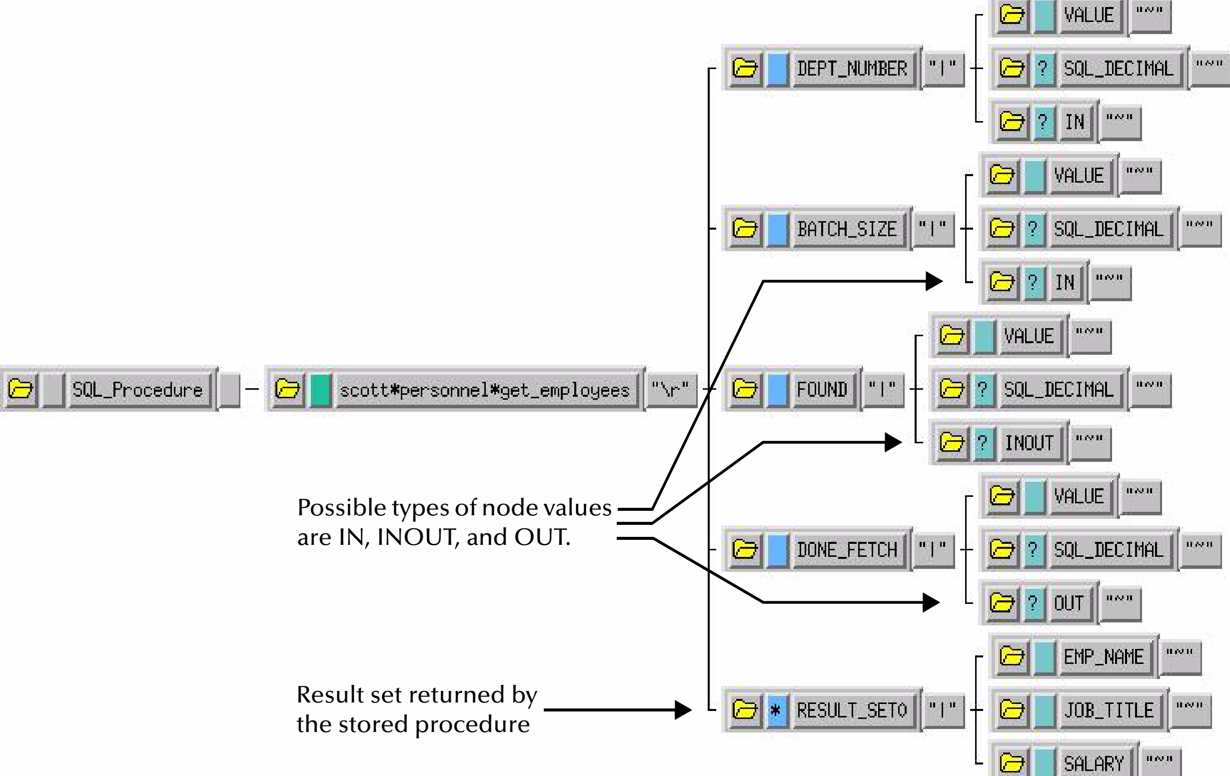
The stored procedure specified generates an the ETD shown in Figure 18. Below is the contents of the sample stored procedure:

```
procedure GET_EMPLOYEES
(
   dept_number in     integer,
   batch_size  in     integer,
   found       in out integer,
   done_fetch  out    integer,
   emp_name    out    charArrayTyp,
   job_title   out    charArrayTyp,
   salary      out    numArrayTyp
) is
begin
   if not get_emp%isopen then
      open get_emp(dept_number);
   end if;
   done_fetch := 0;
   found := 0;
   for i in 1..batch_size loop
      fetch get_emp into emp_name(i),
         job_title(i), salary(i);
      if get_emp%notfound then
         close get_emp;
         done_fetch := 1;
         exit;
      else
         found := found + 1;
      end if;
   end loop;
end get_employees;
```

Note: *The stored procedure shown above uses the PL/SQL table (array) type that is unique to Oracle stored procedures. The output parameters **emp_name**, **job_title**, and **salary** are returned as an array. These parameters are represented in the generated ETD as a "result set." See* **Figure 18 on page 53** *for an example of a result set.*

Note: *Although periods can be entered in the selection criteria in the Build Tool, they are not permitted in the node names of the ETD. Any periods in the selection criteria will be converted to asterisks in the generated ETD. See* **Figure 18 on page 53**.

Figure 18 Stored Procedure ETD



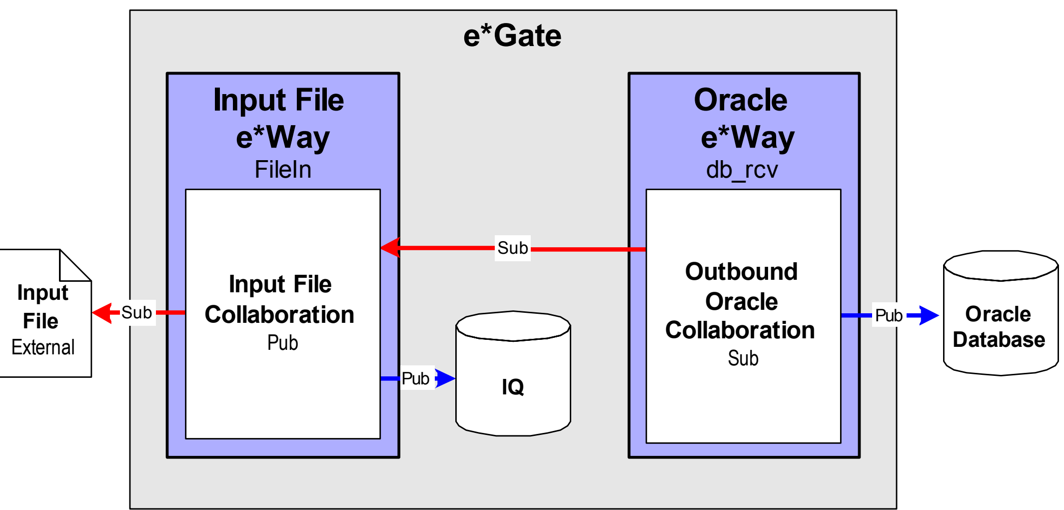
This Event Type Definition is used to pass certain input to the stored procedure. The nodes with types of **IN** or **INOUT** are used as input. The nodes with types of **OUT** or **INOUT** can be used for output. The results of the stored procedure are returned to the **RESULT_SET0** node. The Build Tool will create additional result set nodes (**RESULT_SET1**, **RESULT_SET2**, and so forth) for stored procedures returning multiple results.

4.2 Sample One—Publishing e*Gate Events to an Oracle Database

This section describes how to use the Oracle e*Way in a sample implementation. This sample schema demonstrates the publishing of e*Gate Events to an Oracle database.

This scenario uses a file e*Way to load an input file containing employee information and generate the initial Event. The Oracle e*Way subscribes to the Event and inserts the employee records into the external Oracle database.

Figure 19 Publishing to Oracle database

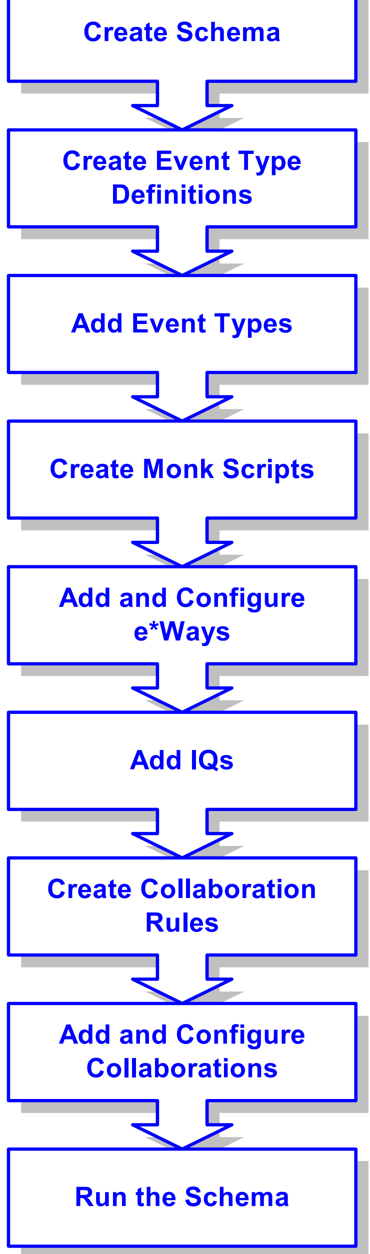


Overview of Steps

The sample implementation follows these general steps:

- **“Create the Schema” on page 55**
- **“Create the Event Type Definitions” on page 56**
- **“Add the Event Types” on page 57**
- **“Create the Monk Scripts” on page 58**
- **“Add and Configure the e*Ways” on page 59**
- **“Add the IQs” on page 61**
- **“Create the Collaboration Rules” on page 61**
- **“Add and Configure the Collaborations” on page 62**
- **“Run the Schema” on page 63**

Figure 20 Schema Configuration Steps



4.2.1 Create the Schema

The first step in deploying the sample implementation is to create a new Schema. After installing the Oracle e*Way Intelligent Adapter, do the following:

1. Launch the e*Gate Schema Designer GUI.
2. When the Schema Designer prompts you to log in, select the Registry Host, User Name, and Password to be used to log in and click **Open**.

3 From the list of Schemas, click **New** to create a new Schema.

4 For this sample implementation, enter the name **Oracle_Sample1** and click **Open**.

The Schema Designer will launch and display the newly created Schema.

4.2.2 Create the Event Type Definitions

Three Event Type Definitions are used in this sample. The ETDs are:

- **EventMsg.ssc** – This standard ETD is used by the **FileInEvent** Event Type.
- **db_rcv_in.ssc** – This user-created ETD contains basic employee information such as name, rate, and date.
- **db_rcv_struct.ssc** – This user-created ETD contains the same basic employee information formatted appropriately for the Oracle data source.

To create the db_rcv_in ETD:

1 From the e*Gate Schema Designer, click to launch the ETD Editor.

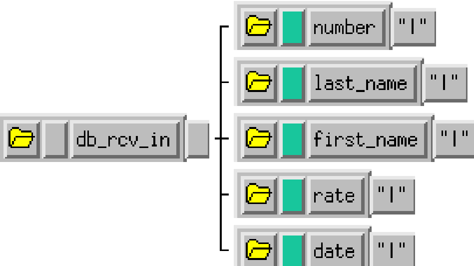
2 Click to create the new ETD.

The New ETD dialog will be displayed.

3 Enter **db_rcv_in.ssc** as the file name for the ETD.

4 Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 21 The **db_rcv_in.ssc** ETD



5 Click to save the ETD.

6 From the **File** menu, select **Promote to Run Time**. Click **Yes** to confirm the promotion of the file.

To create the db_rcv_struct ETD:

1 From the e*Gate Schema Designer, click to launch the ETD Editor.

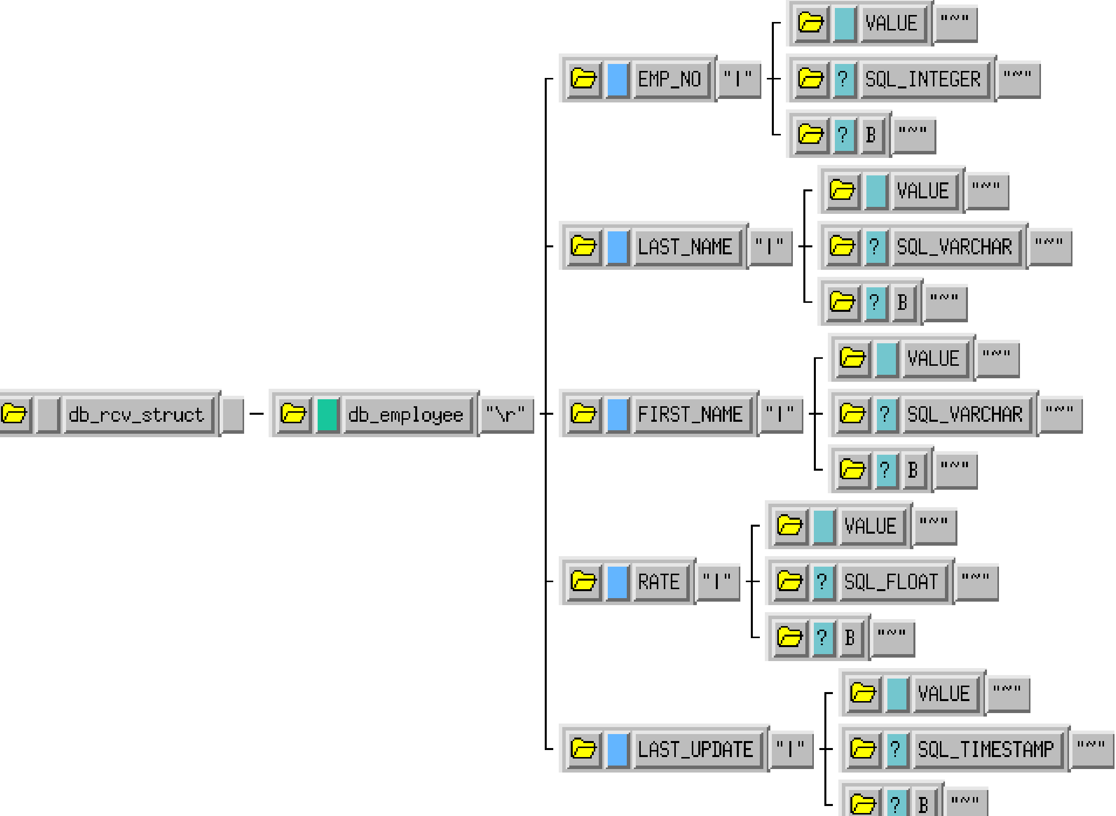
2 Click to create the new ETD.

The New ETD dialog will be displayed.

3 Enter **db_rcv_struct.ssc** as the file name for the ETD.

4 Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 22 The **db_rcv_struct.ssc** ETD



5 Click Save to save the ETD.

6 From the **File** menu, select **Promote to Run Time**. Click **Yes** to confirm the promotion of the file.

4.2.3 Add the Event Types

Three Event Types are used in this sample. The Event Types are:

- **FileInEvent** – This Event Type represents the inbound data from an external input file. This Event Type uses the **EventMsg.ssc** ETD.
- **db_rcv_in** – This Event Type represents the data transported by the input file e*Way. This Event Type uses the **db_rcv_in.ssc** ETD.
- **db_rcv_struct** – This Event Type represents the transformed Event that will be written to the external Oracle database. This Event Type uses the **db_rcv_struct.ssc** ETD.

To add the Event Types:

1 In the components pane of the Schema Designer, select the Event Types folder.

2 Click to add a new Event Type.

3 Enter **FileInEvent** and click **OK**.

4 Select the newly created Event Type and click to display the Event Type's properties.

5 Click **Find** to display the list of Event Types.

6 Navigate to the **monk_scripts\common** folder, select **EventMsg.ssc**, and click **Select**.

7 Click **OK** to close the Event Type's properties.

Repeat these steps for the **db_rcv_in** and **db_rcv_struct** Event Types using the appropriate Event Type Definition files.

4.2.4 Create the Monk Scripts

This sample implementation uses a DART script (**db_rcv.dsc**) to communicate with the external Oracle database.

To create the DART script:

1 From the e*Gate Schema Designer, click to launch the Collaboration Rules Editor.

2 Click to create a new DART script.

The New Collaboration Rules Script dialog will be displayed.

3 Enter the name **db_rcv** (with no file extension) as the **File name**.

4 Select **DART Send** from the list of file types. The extension **.dsc** will be appended to the file name.

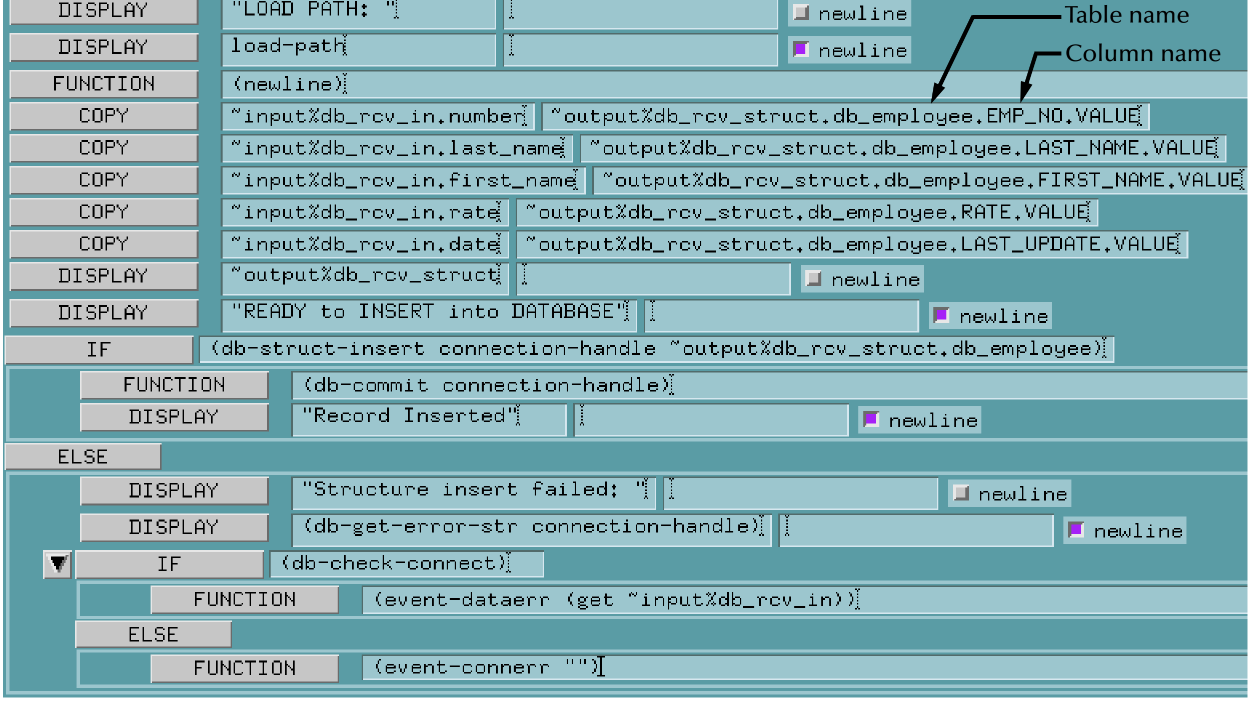
5 Click to display the list of source files. Select **db_rcv_in.ssc** as the source file.

6 Click to display the list of destination files. Select **db_rcv_struct.ssc** as the destination file.

7 Enter the rules as shown in Figure 23.

Note: *The rules shown in Figure 23 use a table named **db_employee**. In order for this sample to work correctly, you must either create a table in your Oracle database called **db_employee** or change each of the references to the table name in your DART script rules as appropriate.*

Figure 23 The **db_rcv.dsc** DART script



8 Click to save the script.

9 Close the Collaboration Rules Script Editor.

4.2.5 Add and Configure the e*Ways

The sample Schema uses two e*Ways: **FileIn** and **Oracle_rcv**. The **FileIn** e*Way reads in the input data file and queues it for the Oracle e*Way. The **Oracle_rcv** e*Way writes the records to the **db_employee** table in the Oracle database.

To add and configure the FileIn e*Way:

1 In the components pane of the Schema Designer, select the Control Broker and click to add a new e*Way.

2 Enter **FileIn** for the component name and click **OK**.

3 Select the newly created e*Way and click to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 4.

Table 4 FileIn e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	default
Outbound (send) settings	All	default
Poller (inbound) settings	PollDirectory	c:\egate\data\dart
	InputFileMask	*.dat
	All others	default
Performance Testing	All	default

7 Select **Save** from the **File** menu. Enter **FileIn** as the file name and click **Save**.

8 Select **Promote to Run Time** from the **File** menu. Click **OK** to continue.

9 A message will notify you that the file has been promoted to run time. Click **OK** to close the e*Way configuration file editor.

10 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

11 Click **OK** to save the e*Way properties.

To add and configure the Oracle_rcv e*Way:

1 In the components pane of the Schema Designer, select the Control Broker and click to add a new e*Way.

2 Enter **Oracle_rcv** for the component name and click **OK**.

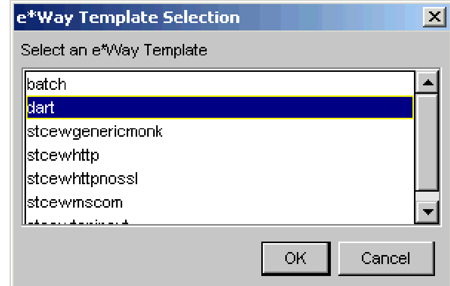
3 Select the newly created e*Way and click to display the e*Way's properties.

4 Use the **Find** button to select **stcewgenericmonk.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Select the **dart** e*Way template and click OK. See Figure 24.

Figure 24 DART e*Way Template Selection



7 Enter the parameters for the e*Way as shown in Table 5.

Table 5 Oracle_rcv e*Way Parameters

Section Name	Parameter	Value
General Settings	All	default
Communication Setup	Start Exchange Data Schedule	Repeatedly, every 1 minute
	All others	default
Monk Configuration	Process Outgoing Message Function	monk_scripts\common\db_rcv.dsc
	Exchange Data With External Function	monk_scripts\common\db_rcv.dsc
	All others	default
Database Setup	Database Type	Oracle8 ORACLE8i ORACLE9i
	All others	Use local settings

Note: *Use the appropriate **Database Name**, **User Name**, and **Encrypted Password** according to your local Oracle implementation.*

8 Save the e*Way's configuration file and promote it to run time.

9 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

10 Click **OK** to save the e*Way properties.

4.2.6 Add the IQs

The sample Schema requires one Intelligent Queue—**OracleIQ**.

To add the IQ:

1 In the components pane of the Schema Designer, select the IQ manager. Click [icon] to create the new IQ.

2 Enter the name **OracleIQ** and click **OK** to save the IQ.

3 Select the IQ Manager and click [icon] to display the IQ Manager's properties.

4 In the **Start Up** tab of the IQ Manager's properties, select the **Start automatically** check box.

5 Click **OK** to save the IQ Manager's properties.

4.2.7 Create the Collaboration Rules

This sample schema uses two Collaboration Rules:

- **InboundEvent** – This Collaboration Rule is used by the **FileIn** e*Way's collaboration to transform the **FileInEvent** Events into **db_rcv_in** Events.
- **OutboundEvent** – This Collaboration Rule is used by the **Oracle_rcv** e*Way's collaboration to transform the **db_rcv_in** Events into **db_rcv_struct** Events.

To add the InboundEvent Collaboration Rule:

1. In the components pane of the Schema Designer, select the Collaboration Rules folder.
2. Click the button to create a new Collaboration Rule.
3. Enter the name **InboundEvent** and click **OK**.
4. Select the newly created Collaboration Rule and click to display the Collaboration Rule's properties.
5. In the **General** tab, select the **Pass Through** service.
6. Under the Subscriptions tab, select the **FileInEvent** Event Type.
7. Under the Publications tab, select the **db_rcv_in** Event Type.
8. Click **OK** to save and close the Collaboration Rule.

To add the OutboundEvent Collaboration Rule:

1. In the components pane of the Schema Designer, select the Collaboration Rules folder.
2. Click the button to create a new Collaboration Rule.
3. Enter the name **OutboundEvent** and click **OK**.
4. Select the newly created Collaboration Rule and click to display the Collaboration Rule's properties.
5. In the **General** tab, select the **Pass Through** service.
6. Under the Subscriptions tab, select the **db_rcv_in** Event Type.
7. Under the Publications tab, select the **db_rcv_struct** Event Type.
8. Click **OK** to save and close the Collaboration Rule.

4.2.8 Add and Configure the Collaborations

Each of the two e*Ways uses one Collaboration to route the Events through the sample Schema.

- **FileIn_collab** – This collaboration is used by the FileIn e*Way to process the inbound Event and queue it for the **Oracle_rcv** e*Way.
- **Oracle_rcv_collab** – This collaboration subscribes to the Event from the **FileIn_collab** and publishes the Event to the Oracle database.

To create the FileIn_collab Collaboration:

1. In the components pane of the Schema Designer, select the **FileIn** e*Way.
2. Click the button to create a new Collaboration.
3. Enter the name **FileIn_collab** and click **OK**.
4. Select the newly created Collaboration and click to display the Collaboration's properties.
5. Select **InboundEvent** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **FileInEvent** Event Type and the **<External>** source.
8. Click **Add** to add a new Publication.
9. Select the **db_rcv_in** Event Type and the **OracleIQ** destination.
10. Click **OK** to close the Collaboration's properties.

To create the Oracle_rcv_collab Collaboration:

1. In the components pane of the Schema Designer, select the **Oracle_rcv** e*Way.
2. Click the button to create a new Collaboration.
3. Enter the name **Oracle_rcv_collab** and click **OK**.
4. Select the newly created Collaboration and click to display the Collaboration's properties.
5. Select **OutboundEvent** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **db_rcv_in** Event Type and the **FileIn_collab** source.
8. Click **Add** to add a new Publication.
9. Select the **db_rcv_struct** Event Type and the **<External>** destination.
10. Click **OK** to close the Collaboration's properties.

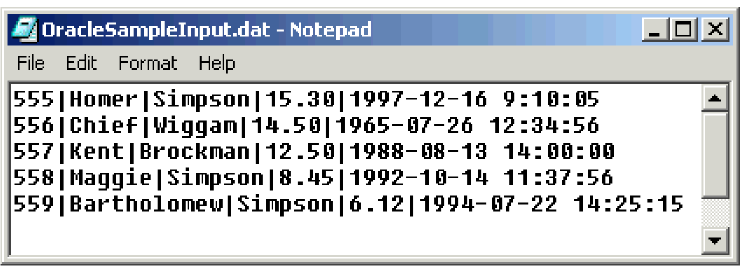
4.2.9 Run the Schema

Running the sample Schema requires a sample input file to be created. Once the input file has been created, you can start the Control Broker from a command prompt to execute the Schema. After the Schema has been run, you can use a query utility to query the results in the Oracle database.

The sample input file

Use a text editor to create an input file to be read by the inbound file e*Way (**FileIn**). The file must be formatted to match the ETD used by the DART script (see **Figure 21 on page 56**). An example of an input file is shown in Figure 25. Save the file to the directory specified in the e*Way's configuration file (such as **c:\egate\data\dart**).

Figure 25 Sample Input File



To run the Control Broker:

From a command line, type the following command:

```
stccb -ln logical_name -rh registry -rs schema -un user_name -up
password
```

where

logical_name is the logical name of the Control Broker,

registry is the name of the Registry Host,

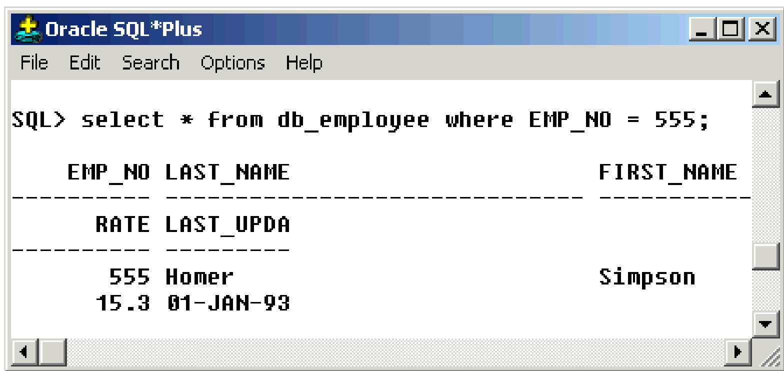
schema is the name of the Registry Schema, and

user_name and *password* are a valid e*Gate username/password combination.

To verify the results:

Use an SQL query utility (such as Oracle SQL Plus) to query the results of the output to the Oracle database. Figure 26 shows an example of a query to verify the results of the schema's output based on the input file used by this example.

Figure 26 Sample Output Console

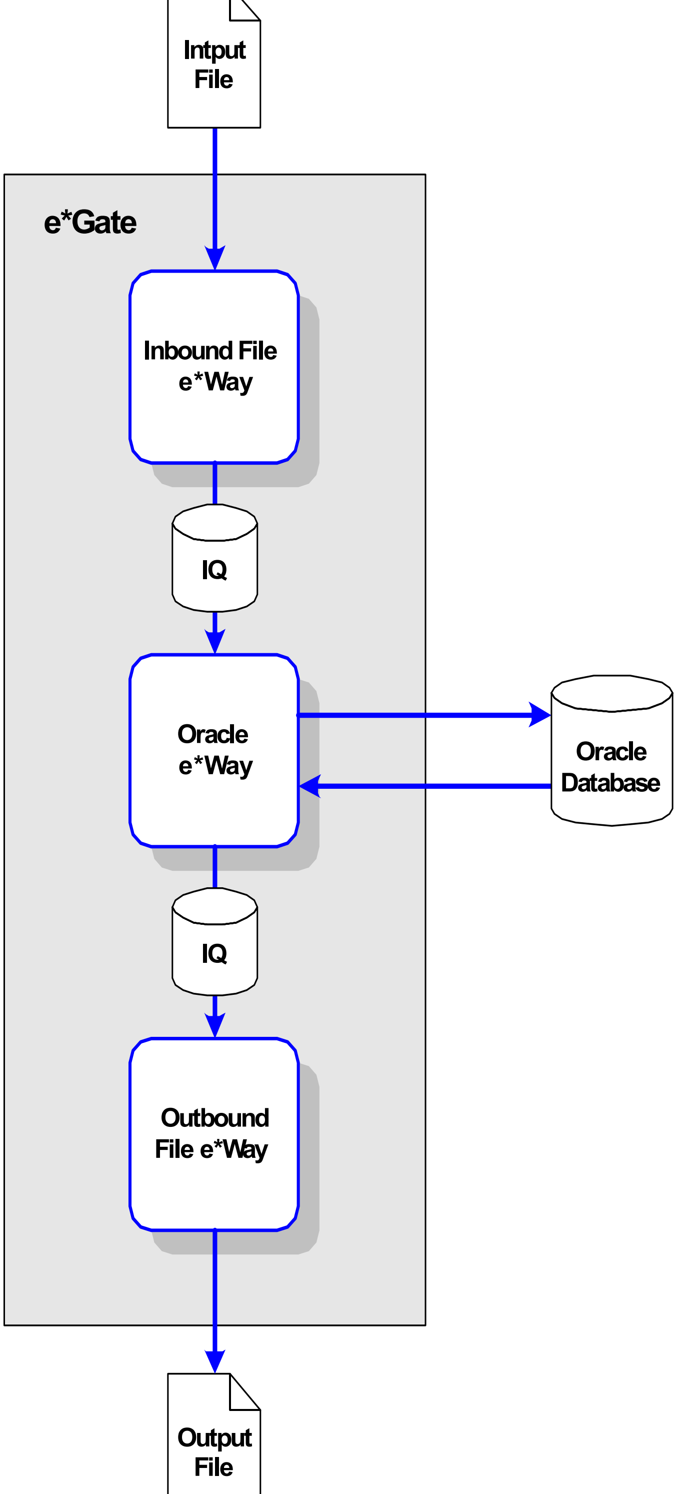


4.3 Sample Two—Polling from an Oracle Database

This section describes how to use the Oracle e*Way in a sample implementation. This sample schema demonstrates the polling of records from an Oracle database and converting the records into e*Gate Events.

The scenario uses a file e*Way to load an input file containing employee numbers. These employee numbers are used to converted into e*Gate Events. The Oracle e*Way uses these inbound Events to poll employee records from the external Oracle database. As the records are returned to the Oracle e*Way, the Events are published to the outbound IQ. The Outbound file e*Way finally writes the employee records to the output file.

Figure 27 Polling from Oracle Database



Overview of Steps

This sample implementation follows these general steps:

- **"Create the Schema" on page 67**
- **"Create the Event Type Definitions" on page 67**
- **"Add the Event Types" on page 68**
- **"Create the Monk Scripts" on page 69**
- **"Add and Configure the e*Ways" on page 71**
- **"Add the IQs" on page 73**
- **"Create the Collaboration Rules" on page 74**
- **"Add and Configure the Collaborations" on page 74**
- **"Run the Schema" on page 76**

Note: *The procedures outlined in this sample are not explained in the same level of detail as in* **Sample One—Publishing e*Gate Events to an Oracle Database** *on page 53. For additional information regarding the configuration of e*Gate components, see* ***Creating an End-to-End Scenario with e*Gate Integrator****.*

4.3.1 Create the Schema

The first step in deploying this sample implementation is to create a new Schema.

To add the new Schema:

1. Log into the e*Gate Schema Designer.
2. When you are prompted to select a Schema, click **New** to add a new Schema.
3. Name the Schema **Oracle_Sample2**.

4.3.2 Create the Event Type Definitions

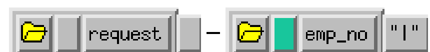
The sample scenario requires two Event Type Definitions. The ETDs are:

- **db_request.ssc** – This ETD is used to format the inbound request Events.
- **db_reply.ssc** – This ETD is used to format the outbound reply Events.

To create the db_request ETD:

1. From the e*Gate Schema Designer, click [icon] to launch the ETD Editor.
2. Create a new ETD named **db_request.ssc**.
3. Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 28 The **db_request.ssc** ETD



4 Save the ETD and promote it to Run Time.

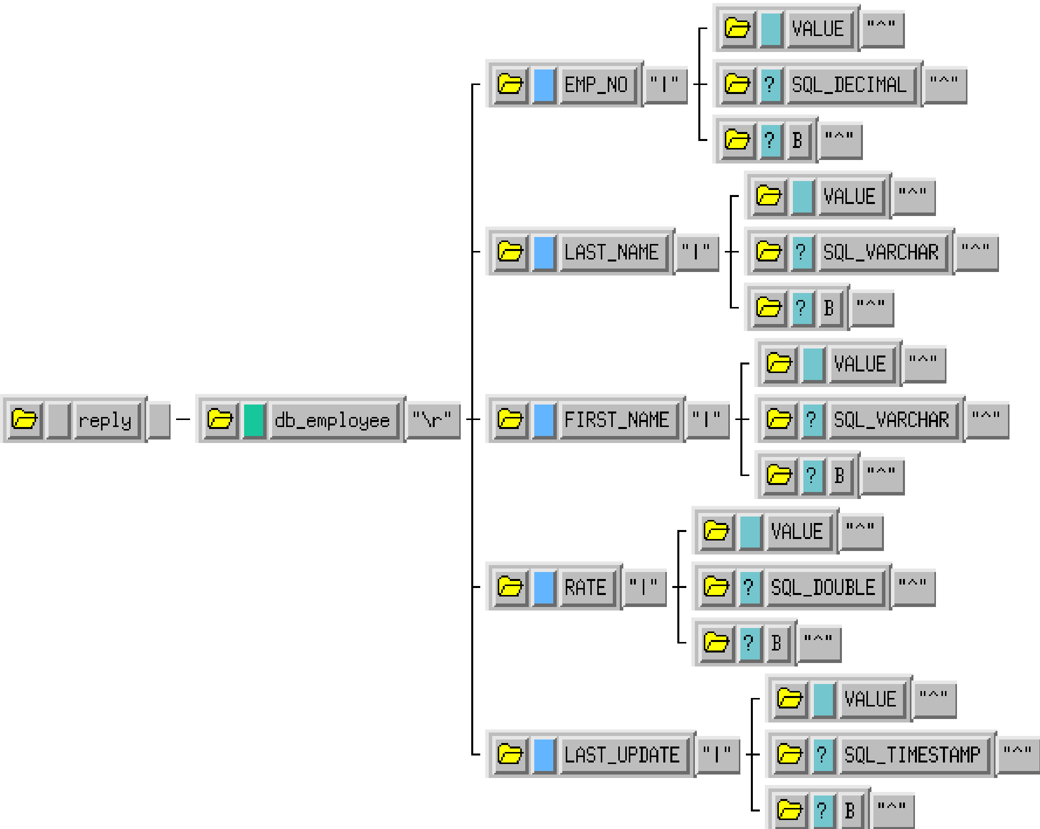
To create the db_reply ETD:

1 From the e*Gate Schema Designer, click [icon] to launch the ETD Editor.

2 Create a new ETD named **db_reply.ssc**.

3 Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 29 The **db_reply.ssc** ETD



4 Save the ETD and promote it to Run Time.

4.3.3 Add the Event Types

The sample scenario requires six Event Types. The Event Types are:

- **InboundFile** – This Event Type represents the inbound file as it is loaded from the file system.
- **InboundEvent** – This Event Type represents the inbound record that has been converted to an e*Gate Event.
- **PollRequest** – This Event Type represents the request that is sent to the Oracle database.

- **PollReply** – This Event Type represents the reply that is returned by the Oracle database.
- **OutboundEvent** – This Event Type represents the outbound Event to be sent to the external file system.

4.3.4 Create the Monk Scripts

This sample implementation uses a DART script (**db_poll.dsc**) to poll the external Oracle database.

To create the DART script:

1. From the e*Gate Schema Designer, click to launch the Collaboration Rules Editor.
2. Click to create a new DART script.

   The New Collaboration Rules Script dialog will be displayed.
3. Enter the name **db_poll** (with no file extension) as the **File name**.
4. Select **DART Poll** from the list of file types. The extension **.dsc** will be appended to the file name.
5. Click to display the list of source files. Select **db_request.ssc** as the source file.
6. Click to display the list of destination files. Select **db_struct.ssc** as the destination file.
7. Enter the rules as shown in Figure 30.

Note: *The rules shown in Figure 30 use a table named **db_employee**. In order for this sample to work correctly, you must either create a table in your Oracle database called **db_employee** or change each of the references to the table name in your DART script rules as appropriate.*

Figure 30 The **db_poll.dsc** DART script

8 Click [Save] to save the script.

9 Close the Collaboration Rules Script Editor.

4.3.5 Add and Configure the e*Ways

The sample Schema uses three e*Ways:

- **FileIn** – The **FIleIn** e*Way reads in the input data file and queues it for the Oracle e*Way.
- **Oracle** – The **Oracle** e*Way polls the db_employee table in the Oracle database and queues the returned data for the outbound file e*Way.
- **FileOut** – The **FileOut** e*Way writes the records returned by the Oracle e*Way to the output text file.

To add and configure the FileIn e*Way:

1 In the components pane of the Schema Designer, select the Control Broker and click to add a new e*Way.

2 Enter **FileIn** for the component name and click **OK**.

3 Select the newly created e*Way and click to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 6.

Table 6 FileIn e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	Yes
	AllowOutgoing	No
	Performance Testing	default
Outbound (send) settings	All settings	default
Poller (inbound) settings	PollDirectory	c:\egate\data\dart
	OutputFileName	*.dat
	AllOthers	default
Performance Testing	All settings	default

7 Save the e*Way's configuration file and promote it to run time.

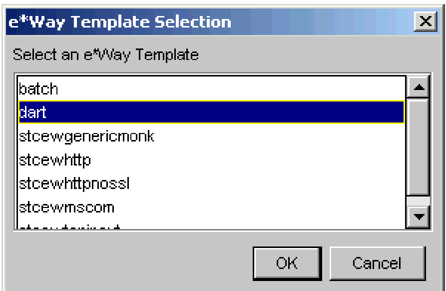
8 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

9 Click **OK** to save the e*Way properties.

To add and configure the Oracle e*Way:

1. In the components pane of the Schema Designer, select the Control Broker and click to add a new e*Way.
2. Enter **Oracle** for the component name and click **OK**.
3. Select the newly created e*Way and click to display the e*Way's properties.
4. Use the **Find** button to select **stcewgenericmonk.exe** as the executable file.
5. Click **New** to create a new configuration file.
6. Select the **dart** e*Way template and click OK. See Figure 31.

Figure 31 DART e*Way Template Selection



7. Enter the parameters for the e*Way as shown in Table 7.

Table 7 Oracle e*Way Parameters

Section Name	Parameter	Value
General Settings	All	default
Communication Setup	Start Exchange Data Schedule	Repeatedly, 30 seconds
	All others	default
Monk Configuration	Process Outgoing Message Function	monk_scripts\common\db_poll.dsc
	All others	default
Database Setup	Database Type	ORACLE8 ORACLE8i ORACLE9i
	All others	Use local settings

Note: *Use the appropriate **Database Name**, **User Name**, and **Encrypted Password** according to your local Oracle implementation.*

8. Save the e*Way's configuration file and promote it to run time.
9. In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.
10. Click **OK** to save the e*Way properties.

To add and configure the FileIn e*Way:

1. In the components pane of the Schema Designer, select the Control Broker and click to add a new e*Way.
2. Enter **FileOut** for the component name and click **OK**.
3. Select the newly created e*Way and click to display the e*Way's properties.
4. Use the **Find** button to select **stcewfile.exe** as the executable file.
5. Click **New** to create a new configuration file.
6. Enter the parameters for the e*Way as shown in Table 6.

Table 8 FileOut e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
	Performance Testing	default
Outbound (send) settings	OutputDirectory	c:\egate\data\dart
	OutputFileName	PollOutput%d.dat
	All Others	default
Poller (inbound) settings	All	default
Performance Testing	All	default

7. Save the e*Way's configuration file and promote it to run time.
8. In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.
9. Click **OK** to save the e*Way properties.

4.3.6 Add the IQs

This sample Schema requires two Intelligent Queues: Oracle1IQ and Oracle2IQ.

To add the IQs:

1. In the components pane of the Schema Designer, select the IQ manager. Click to create the first new IQ.
2. Enter the name **Oracle1IQ** and click **Apply** to save the first IQ.
3. Enter the name **Oracle2IQ** and click **OK** to save the second IQ.
4. Select the IQ Manager and click to display the IQ Manager's properties.
5. In the **Start Up** tab of the IQ Manager's properties, select the **Start automatically** check box.
6. Click **OK** to save the IQ Manager's properties.

4.3.7 Create the Collaboration Rules

This sample schema uses four Collaboration Rules:

- **FileIn** – This Collaboration Rule is used by the **FileIn** e*Way's Collaboration to transform the **InboundFile** Events into **InboundEvent** Events.
- **OracleRequest** – This Collaboration Rule is used by the **Oracle** e*Way's Collaboration to transform the **InboundEvent** Events into **PollRequest** Events.
- **OracleReply** – This Collaboration Rule is used by the **Oracle** e*Way's Collaboration to transform the **PollRequest** Events into **PollReply** Events.
- **FileOut** – This Collaboration Rule is used by the **FileOut** e*Way's Collaboration to transform the **PollReply** Events into **OutboundEvent** Events.

To add the FileIn Collaboration Rule:

1. In the components pane of the Schema Designer, select the Collaboration Rules folder.
2. Click the button to create a new Collaboration Rule.
3. Enter the name **FileIn** and click **OK**.
4. Select the newly created Collaboration Rule and click to display the Collaboration Rule's properties.
5. In the **General** tab, select the **Pass Through** service.
6. Under the Subscriptions tab, select the **InboundFile** Event Type.
7. Under the Publications tab, select the **InboundEvent** Event Type.
8. Click **OK** to save and close the Collaboration Rule.

To add the remaining Collaboration Rules:

Follow the same steps used to add the **FileIn** Collaboration Rule using the names and Event Types shown at the beginning of this section.

4.3.8 Add and Configure the Collaborations

This sample schema uses four Collaborations:

- **FileIn_collab** – This Collaboration is used to transform the **InboundFile** Events into **InboundEvent** Events.
- **OracleRequest_collab** – This Collaboration is used to transform the **InboundEvent** Events into **PollRequest** Events.
- **OracleReply_collab** – This Collaboration is used to transform the **PollRequest** Events into **PollReply** Events.
- **FileOut_collab** – This Collaboration is used to transform the **PollReply** Events into **OutboundEvent** Events.

To create the FileIn_collab Collaboration:

1. In the components pane of the Schema Designer, select the **FileIn** e*Way.
2. Click the button to create a new Collaboration.
3. Enter the name **FileIn_collab** and click **OK**.
4. Select the newly created Collaboration and click to display the Collaboration's properties.
5. Select **InboundFile** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **InboundEvent** Event Type and the **<External>** source.
8. Click **Add** to add a new Publication.
9. Select the **InboundEvent** Event Type and the **Oracle1IQ** destination.
10. Click **OK** to close the Collaboration's properties.

To create the OracleRequest_collab Collaboration:

1. In the components pane of the Schema Designer, select the **Oracle** e*Way.
2. Click the button to create a new Collaboration.
3. Enter the name **OracleRequest_collab** and click **OK**.
4. Select the newly created Collaboration and click to display the Collaboration's properties.
5. Select **OracleRequest** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **InboundFile** Event Type and the **FileIn_Collab** source.
8. Click **Add** to add a new Publication.
9. Select the **OracleRequest** Event Type and the **<External>** destination.
10. Click **OK** to close the Collaboration's properties.

To create the OracleReply_collab Collaboration:

1. In the components pane of the Schema Designer, select the **Oracle** e*Way.
2. Click the button to create a new Collaboration.
3. Enter the name **OracleReply_collab** and click **OK**.
4. Select the newly created Collaboration and click to display the Collaboration's properties.
5. Select **OracleRereply** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **OracleRequest** Event Type and the **<External>** source.

8. Click **Add** to add a new Publication.
9. Select the **OracleReply** Event Type and the **Oracle2IQ** destination.
10. Click **OK** to close the Collaboration's properties.

To create the FileOut_collab Collaboration:

1. In the components pane of the Schema Designer, select the **FileOut** e*Way.
2. Click the button to create a new Collaboration.
3. Enter the name **FileOut_collab** and click **OK**.
4. Select the newly created Collaboration and click to display the Collaboration's properties.
5. Select **FileOut** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **OracleReply** Event Type and the **OracleReply_collab** source.
8. Click **Add** to add a new Publication.
9. Select the **OutboundEvent** Event Type and the **<External>** destination.
10. Click **OK** to close the Collaboration's properties.

4.3.9 Run the Schema

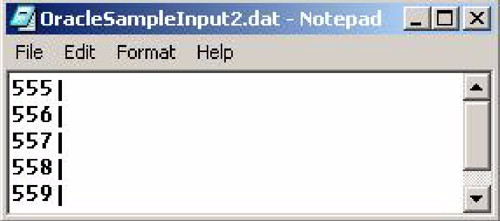
Running the sample Schema requires a sample input file to be created. Once the input file has been created, you can start the Control Broker from a command prompt to execute the Schema. After the Schema has been run, you can view the results in the output file.

The sample input file

Use a text editor to create an input file to be ready by the inbound file e*Way (**FileIn**). The file must be formatted to match the simple ETD used by the DART script (see **Figure 28 on page 67**). An example of an input file is shown in Figure 32. Save the file to the directory specified in the e*Way's configuration file (such as **c:\egate\data\dart**).

Note: *The "employee numbers" used in this example must exist in your Oracle database. The sample shown below uses employee numbers that exist from the records in the previous sample schema.*

Figure 32 Sample Input File



To run the Control Broker:

From a command line, type the following command:

```
stccb -ln logical_name -rh registry -rs OracleSample2 -un user_name
-up password
```

where

logical_name is the logical name of the Control Broker,

registry is the name of the Registry Host, and

user_name and ***password*** are a valid e*Gate username/password combination.

To verify the results:

Use a text editor to view the records that were written to the output file specified by the FileOut e*Way. The records should correspond to the records in the external database.

Chapter 5

Oracle e*Way Functions

The functions described in this chapter control the Oracle e*Way's basic operations as well as those needed for database access.

This Chapter Explains:

- **"Basic Functions" on page 78**
- **"Standard e*Way Functions" on page 86**
- **"General Connection Functions" on page 104**
- **"Static SQL Functions" on page 119**
- **"Dynamic SQL Functions" on page 135**
- **"Stored Procedure Functions" on page 149**
- **"Message Event Functions" on page 180**
- **"Sample Monk Scripts" on page 192**

5.1 Basic Functions

The functions in this category control the e*Way's most basic operations.

The basic functions are:

event-send-to-egate on page 79

get-logical-name on page 80

send-external-down *on page 81*

send-external-up *on page 82*

shutdown-request on page 83

start-schedule on page 84

stop-schedule on page 85

event-send-to-egate

Syntax

```
(event-send-to-egate string)
```

Description

event-send-to-egate sends an Event from the e*Way. If the external collaboration(s) is successful in publishing the Event to the outbound queue, the function will return **#t**, otherwise **#f**.

Parameters

Name	Type	Description
string	string	The data to be sent to the e*Gate system

Return Values

Boolean
Returns **#t** when successful and **#f** when an error occurs.

Throws

None.

Additional information

This function can be called by any e*Way function when it is necessary to send data to the e*Gate system in a blocking fashion.

get-logical-name

Syntax

```
(get-logical-name)
```

Description

get-logical-name returns the logical name of the e*Way.

Parameters

None.

Return Values

string
Returns the name of the e*Way (as defined by the Schema Designer).

Throws

None.

send-external-down

Syntax

```
(send-external-down)
```

Description

send-external down instructs the e*Way that the connection to the external system is down.

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

```
(send-external-up)
```

Description

send-external-up instructs the e*Way that the connection to the external system is up.

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown-request completes the e*Gate shutdown procedure that was initiated by the Control Broker but was interrupted by returning a non-null value within the **Shutdown Command Notification Function** (see **"Shutdown Command Notification Function" on page 41**). Once this function is called, shutdown proceeds immediately.

Once interrupted, the e*Way's shutdown cannot proceed until this Monk function is called. If you do interrupt an e*Way shutdown, we recommend that you complete the process in a timely fashion.

Parameters

None.

Return Values

None.

Throws

None.

start-schedule

Syntax

```
(start-schedule)
```

Description

start-schedule requests that the e*Way execute the “Exchange Data with External” function specified within the e*Way’s configuration file. Does not effect any defined schedules.

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

stop-schedule requests that the e*Way halt execution of the “Exchange Data with External” function specified within the e*Way’s configuration file. Execution will be stopped when the e*Way concludes any open transaction. Does not effect any defined schedules, and does not halt the e*Way process itself.

Parameters

None.

Return Values

None.

Throws

None.

5.2 Standard e*Way Functions

The functions in this category control the e*Way's standard operations.

The standard functions are:

db-stdver-conn-estab on page 87

db-stdver-conn-shutdown on page 89

db-stdver-conn-ver on page 90

db-stdver-data-exchg on page 92

db-stdver-data-exchg-stub on page 93

db-stdver-init on page 94

db-stdver-neg-ack on page 96

db-stdver-pos-ack on page 97

db-stdver-proc-outgoing on page 98

db-stdver-proc-outgoing-stub on page 100

db-stdver-shutdown on page 102

db-stdver-startup on page 103

db-stdver-conn-estab

Syntax

```
(db-stdver-conn-estab)
```

Description

db-stdver-conn-estab is used to establish external system connection.The following tasks are performed by this function:

- construct a new connection handle
- call db-long to connect to database
- setup timestamp format if required
- setup maximum long data buffer limit if required
- bind dynamic SQL statement and stored procedures.

Parameters

None.

Return Values

A string
UP or **SUCCESS** if connection established, anything else if connection not established.

Throws

None.

Additional Information

To use standard database time format, add the following function call to this function: (**db-std-timestamp-format** *connection-handle*) after the (**db-bind**) call.

For "Maximum Long Data Size" the DART library allocates an internal buffer for each SQL_LONGVARCHAR and SQL_LONGVARBINARY data, when the SQL statement or stored procedure that contains these data types are bound. The default size of each internal data buffer is 1024K(1048576) bytes. If the user needs to handle long data larger than this default value, add the following function call to specify the maximum data size:

```
(db-max-long-data-size connection-handle maximum-data-size)
```

(see **db-max-long-data-size** on page 114 for more information.

Standard Implementation

```
(define db-stdver-conn-estab
 (lambda (  )
  (let ((result "DOWN")(last_dberr ""))
   (display "[++] Executing e*Way external connection establishment
function.")
   (display "db-stdver-conn-estab: logging into the database
with:\n")
   (display "DATABASE NAME = ")
   (display DATABASE_SETUP_DATABASE_NAME)
   (newline )
   (display "USER NAME = ")
   (display DATABASE_SETUP_USER_NAME)
```

```
      (newline )
      (set! connection-handle (make-connection-handle))
      (if (connection-handle? connection-handle)
       (begin
        (if (db-login connection-handle
             DATABASE_SETUP_DATABASE_NAME
             DATABASE_SETUP_USER_NAME
             DATABASE_SETUP_ENCRYPTED_PASSWORD)
         (begin
          (db-std-timestamp-format connection-handle)
          (db-bind )
          (set! result "UP")
         )
         (begin
          (set! last_dberr (db-get-error-str connection-handle))
          (display last_dberr)
          (event-send "ALERTCAT_OPERATIONAL"
                      "ALERTSUBCAT_CANTCONN"
                      "ALERTINFO_FATAL"
                      "0" "Cannot connect to database"
                      (string-append
                           "Failed to connect to database: "
                           DATABASE_SETUP_DATABASE_NAME
                           "with error" last_dberr)
                      0 (list))
          (newline )
          (db-logout connection-handle)
          (set! result "DOWN")
         )
        )
       )
       (begin
        (set! result "DOWN")
        (display "Failed to create connection handle.")
        (event-send "ALERTCAT_OPERATIONAL"
                    "ALERTSUBCAT_UNUSABLE"
                    "ALERTINFO_FATAL"
                    "0"
                    "database connection handle creation error"
                    "Failed to create database connection handle"
                    0 (list))
       )
      )
      result
     )
   ))
```

db-stdver-conn-shutdown

Syntax

```
(db-stdver-conn-shutdown string)
```

Description

db-stdver-conn-shutdown is called by the system to request that the interface disconnect from the external system, preparing for a suspend/reload cycle. Any return value indicates that the suspend can occur immediately, and the interface will be placed in the down state.

Parameters

Name	Type	Description
string	string	When the e*Way calls this function, it will pass the string "SUSPEND_NOTIFICATION" as the parameter.

Return Values

A string

A return of "SUCCESS" indicates that the external is ready to suspend.

Throws

None.

Standard Implementation

```
(define db-stdver-conn-shutdown
 (lambda ( message-string )
  (let ((result "SUCCESS"))
   (display "[++] Executing e*Way external connection shutdown
function.")
   (display message-string)
   (db-logout connection-handle)
   result
  )
))
```

db-stdver-conn-ver

Syntax

```
(db-stdver-conn-ver)
```

Description

db-stdver-conn-ver is used to verify whether external system connection is established.

Parameters

None.

Return Values

A string
UP or **SUCCESS** if connection established, anything else if connection not established.

Throws

None.

Additional Information

To use standard database time format, add the following function call to this function: (**db-std-timestamp-format** connection handle) after the (**db-bind**) call.

Standard Implementation

```
(define db-stdver-conn-ver
 (lambda (  )
  (let ((result "DOWN")(last_dberr ""))
   (display "[++] Executing e*Way external connection verification
function.")
   (display "db-stdver-conn-ver: checking connection status...\n")
   (cond
    ((string=? STCDB "SYBASE")
    (db-sql-select connection-handle "verify" "select getdate()"))
    ((string=? STCDB "ORACLE8i")
    (db-sql-select connection-handle "verify" "select sysdate from
dual"))
    ((string=? STCDB "ORACLE8")
    (db-sql-select connection-handle "verify" "select sysdate from
dual"))
    ((string=? STCDB "ORACLE8")
    (db-sql-select connection-handle "verify" "select sysdate from
dual"))
    (else
     (db-sql-select connection-handle "verify" "select {fn NOW()}"))
   )
   (if (db-alive connection-handle)
    (begin
     (db-sql-fetch-cancel connection-handle "verify")
     (set! result "UP")
    )
    (begin
     (set! last_dberr (db-get-error-str connection-handle))
     (display last_dberr)
     (event-send "ALERTCAT_OPERATIONAL"
                 "ALERTSUBCAT_LOSTCONN"
                 "ALERTINFO_FATAL"
                 "0"
                 "Lost connection to database"
```

```
                        (string-append
                               "Lost connection to database: "
                               DATABASE_SETUP_DATABASE_NAME
                               "with error" last_dberr)
                        0 (list))
            (set! result "DOWN")
           )
          )
          result
         )
        ))
```

db-stdver-data-exchg

Syntax

```
(db-stdver-data-exchg)
```

Description

db-stdver-data-exchg is used for sending a received Event from the external system to e*Gate. The function expects no input.

Parameters

None.

Return Values

A string

An empty string indicates a successful operation. Nothing is sent to e*Gate.

A message-string indicates successful operation and the Event is sent to e*Gate.

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect, on reconnect this function will be re-executed with the same Event.

Throws

None.

Standard Implementation

```
(define db-stdver-data-exchg
 (lambda (  )
  (let ((result ""))
   (display "[++] Executing e*Way external data exchange function.")
   result
  )
))
```

db-stdver-data-exchg-stub

Syntax

```
(db-stdver-data-exchg-stub)
```

Description

db-stdver-data-exchg-stub is used as a place holder for the function entry point for sending an Event from the external system to e*Gate. When the interface is configured as an outbound only connection, this function should not be called. The function expects no input.

Parameters

None.

Return Values

A string

An empty string indicates a successful operation. Nothing is sent to e*Gate.

A message-string indicates a successful operation and the Event is sent to e*Gate.

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect, on reconnect this function will be re-executed with the same input message.

Throws

None.

Standard Implementation

```
(define db-stdver-data-exchg-stub
 (lambda (  )
  (let ((result ""))
   (display "[++] Executing e*Way external data exchange function
stub.")
   (event-send "ALERTCAT_OPERATIONAL"
               "ALERTSUBCAT_INTEREST"
               "ALERTINFO_NONE"
               "0"
               "Possible configuration error."
               "Default eway data exchange function called."
               0 (list))
   result
  )
))
```

db-stdver-init

Syntax

```
(db-stdver-init)
```

Description

db-stdver-init begins the initialization process for the e*Way. The function loads all of the monk extension library files that the other e*Way functions will access.

Parameters

None.

Return Values

A string
If a **FAILURE** string is returned, the e*Way will shutdown. Any other return indicates success.

Throws

None.

Standard Implementation

```
(define db-stdver-init
 (lambda (  )
  (let ((result "SUCCESS"))
   (display "[++] Executing dart e*Way external init function.")
   (display "[++] Loading db-eWay-stdver-funcs.monk  ")
   (display "DATABASE TYPE = ")
   (display DATABASE_SETUP_DATABASE_TYPE)
   (newline )
   (define STCDB DATABASE_SETUP_DATABASE_TYPE)
   (define DART_NULL "_NULL_")
   (define DART_NULL_MODE "INOUT")
   (define STCDATADIR (get-data-dir))
   (define connection-handle 0)
   (if (not (load-extension "stc_monkutils.dll"))
    (begin
     (set! result "FAILURE")
     (display "Failed to load stc_monkutils.dll.")
    )
    (begin
     (display " Loaded stc_monkutils.dll ")
    )
   )
   (if (not (load-extension "stc_dbmonkext.dll"))
    (begin
     (set! result "FAILURE")
     (display "Failed to load stc_dbmonkext.dll.")
     (event-send "ALERTCAT_OPERATIONAL"
                 "ALERTSUBCAT_UNUSABLE"
                 "ALERTINFO_FATAL" "0"
                 "stc_dbmonkext.dllloaderror"
                 "Failedtoloadstc_dbmonkext.dll"
                 0 (list))
    )
    (begin
     (display "Loaded stc_dbmonkext.dll")
    )
   )
```

```
          result
        )
      ))
```

db-stdver-neg-ack

Syntax

```
(db-stdver-neg-ack message-string)
```

Description

db-stdver-neg-ack is used to send a negative acknowledgement to the external system, and for post processing after failing to send data to e*Gate.

Parameters

Name	Description
message-string	The Event for which a negative acknowledgment is sent.

Return Values

A string

An empty string indicates a successful operation.

CONNERR indicates a loss of connection with the external, client moves to a down state and attempts to connect, on reconnect neg-ack function will be re-executed.

Throws

None.

Standard Implementation

```
(define db-stdver-neg-ack
 (lambda ( message-string )
  (let ((result ""))
   (display "[++] Executing e*Way external negative acknowledgement
function.")
   (display message-string)
   result
  )
))
```

db-stdver-pos-ack

Syntax

```
(db-stdver-pos-ack message-string)
```

Description

db-stdver-pos-ack is used to send a positive acknowledgement to the external system, and for post processing after successfully sending data to e*Gate.

Parameters

Name	Description
message-string	The Event for which an acknowledgment is sent.

Return Values

A string

An empty string indicates a successful operation. The e*Way will then be able to proceed with the next request.

CONNERR indicates a loss of connection with the external, client moves to a down state and attempts to connect, on reconnect pos-ack function will be re-executed.

Throws

None.

Standard Implementation

```
(define db-stdver-pos-ack
 (lambda ( message-string )
  (let ((result ""))
   (display "[++] Executing e*Way external positive acknowledgement
function.")
   (display message-string)
   result
  )
))
```

db-stdver-proc-outgoing

Syntax

```
(db-stdver-proc-outgoing message-string)
```

Description

db-stdver-proc-outgoing is used for sending a received message (Event) from e*Gate to the external system.

Parameters

Name	Type	Description
message-string	string	The Event to be processed.

Return Values

A string

An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way will compare the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
- If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.

CONNERR indicates that there is a problem communicating with the external system. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll back the message (Event) to the IQ from which it was obtained.

DATAERR indicates that there is a problem with the message (Event) data itself. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way increments its "failed message (Event)" counter, and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see **Journal File Name** on page 23) the message (Event) will be journaled.

If a string other than the following is returned, the e*Way will create an entry in the log file indicating that an attempt has been made to access an unsupported function.

Throws

None.

Standard Implementation

```
(define db-stdver-proc-outgoing
```

```
 (lambda ( message-string )
  (let ((result ""))
   (display "[++] Executing e*Way external process outgoing message
function.")
   (display message-string)
   result
  )
))
```

db-stdver-proc-outgoing-stub

Syntax

```
(db-stdver-proc-outgoing-stub message-string)
```

Description

db-stdver-proc-outgoing-stub is used as a place holder for the function entry point for sending an Event received from e*Gate to the external system. When the interface is configured as an inbound only connection, this function should not be used. This function is used to catch configuration problems.

Parameters

Name	Type	Description
message-string	string	The Event to be processed.

Return Values

A string

An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way will compare the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
- If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.

CONNERR indicates that there is a problem communicating with the external system. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function** according to the **Down Timeout** schedule, and will roll back the message (Event) to the IQ from which it was obtained.

DATAERR indicates that there is a problem with the message (Event) data itself. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way increments its "failed message (Event)" counter, and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see **Journal File Name** on page 23) the message (Event) will be journaled.

If a string other than the following is returned, the e*Way will create an entry in the log file indicating that an attempt has been made to access an unsupported function.

Throws

None.

Standard Implementation

```
(define db-stdver-proc-outgoing-stub
 (lambda ( message-string )
  (let ((result ""))
   (display "[++] Executing e*Way external process outgoing message
function stub.")
   (display message-string)
   (event-send "ALERTCAT_OPERATIONAL"
               "ALERTSUBCAT_INTEREST"
               "ALERTINFO_NONE"
               "0"
               "Possible configuration error."
               (string-append
                  "Default eway process outgoing msg function "
                  "passed following message: " msg)
               0 (list))
   result
  )
))
```

db-stdver-shutdown

Syntax

```
(db-stdver-shutdown shutdown_notification)
```

Description

db-stdver-shutdown is called by the system to request that the external shutdown. A return value of **SUCCESS** indicates that the shutdown can occur immediately, any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a shutdown-request call from within a Monk function to allow the requested shutdown process to continue.

Parameters

Name	Type	Description
shutdown_notification	string	When the e*Way calls this function, it will pass the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

A string
SUCCESS allows an immediate shutdown to occur, anything else delays shutdown until a **shutdown-request** is executed successfully.

Throws

None.

Standard Implementation

```
(define db-stdver-shutdown
 (lambda ( message-string )
  (let ((result "SUCCESS"))
   (display "[++] Executing e*Way external shutdown command
notification function.")
   result
  )
))
```

db-stdver-startup

Syntax

```
(db-stdver-startup)
```

Description

db-stdver-startup is used for instance specific function loads and invokes setup.

Parameters

None.

Return Values

A string
FAILURE causes shutdown of the e*Way. Any other return indicates success.

Throws

None.

Standard Implementation

```
(define db-stdver-startup
 (lambda ()
  (let ((result "SUCCESS"))
   (display "[++] Executing e*Way external startup function.")
   result
  )
))
```

5.3 General Connection Functions

The functions in this category control the e*Way's database connection operations.

The general connection functions are:

connection-handle? on page 105

db-alive on page 106

db-commit on page 108

db-get-error-str on page 109

db-login on page 111

db-logout on page 113

db-max-long-data-size on page 114

db-rollback on page 115

db-std-timestamp-format on page 116

make-connection-handle on page 117

statement-handle? on page 118

connection-handle?

Syntax

```
(connection-handle? any-variable)
```

Description

connection-handle? determines whether or not the input argument is a *connection-handle* data type.

Parameters

This function requires a single variable of any data type.

Return Values

Boolean

Returns **#t** (true) if the argument is a connection handle; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)
```

Explanation

The above example creates a connection handle called hdbc. An error message is displayed if the newly defined hdbc is not a connection handle.

db-alive

Syntax

```
(db-alive connection-handle)
```

Description

db-alive is used to determine if the cause of a failing operation is due to a broken connection. It returns whether or not the database connection was alive during the last call to any procedure that sends commands to the database server.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean

Returns **#t** (true) if the connection to the database server is still alive; otherwise, returns **#f** (false) if the connection to the database server is either dead or down.Use **db-get-error-str** *t*o retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc "Payroll" "user" "password")
    (begin
        (define sql_statement "select * from person where sex = ‘M’")
        (do ((status #t)) ((not status))
            (if (db-sql-select hdbc "male" sql_statement)
                (begin
                    ...
                )
                (begin
                    (display (db-get-error-str hdbc))
                    (set! status (db-alive hdbc))
                )
            )
        )
        (display "lost database connection !\n"))
        (db-logout hdbc))
    )
)
```

Explanation

The example above illustrates an application that is looking for a certain record in the person table of the “Payroll” database. The function will exit the loop *only* if it loses the connection to the database.

Notes

1 Most functions can detect a dead connection handle except **db-commit** and **db-rollback**. Therefore, when the function returns false, users must check for loss of connection.

2. Once the **db-alive** returns **#f** to indicate either a dead connection handle or an unavailable database server, all the subsequent function calls associated with that connection handle will not be executed, with the exception of **db-logout**. Each of these functions will return false with a “lost database connection” error message.
3. Once it is determined the connection handle is not alive, the only course of action the user can take is to log out from that connection handle, redefine a new connection handle, and try to reconnect to the database.

db-commit

Syntax

```
(db-commit connection-handle)
```

Description

db-commit performs all transactions specified by the connection.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if
    (and
        (db-sql-execute hdbc "delete from employee where first_name =
'John'")
        (db-sql-execute hdbc "update employee set first_name = 'Mary'
where ssn = 123456789")
    )
    (db-commit hdbc)
    (begin
        (display (db-get-error-str hdbc))
        (db-rollback hdbc)
    )
)
...
```

Explanation

This example shows that if the application can successfully delete "John's record" and update "Mary's record" it will commit the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

db-get-error-str

Syntax

```
(db-get-error-str connection-handle)
```

Description

db-get-error-str returns the last error message, and is used when the Oracle e*Way returns a **#f** value.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

A string

A simple error message is returned.

To parse the return error message when it contains an error, use the two standard files that define the error message structure and display the contents of each component of the error message.

```
ORACLE - oramsg.ssc, oramsg_display.monk
```

Throws

None.

Examples

Scenario #1 — sample code for db-get-error-str

```
...
(if (db-sql-execute hdbc "delete from employee" where
first_name='John')
     (db-commit hdbc)
     (display (db-get-error-str hdbc))
)
...
```

Explanation

This example shows that if the application can successfully delete "John's record" it will commit the transaction. Otherwise, the application will print out the error message and roll back the same transaction.

```
(if (db-login hdbc dsn uid pwd)
    (begin
       (display "database login succeed !\n")
       (if (db-sql-execute hdbc "INSERT INTO UNKNOWN VALUES (NULL)")
          (db-commit hdbc)
          (oramsg-display (db-get-error-str hdbc))
       )
       (if (not (db-logout hdbc))
          (oramsg-display (db-get-error-str hdbc))
       )
    )
    (oramsg-display (db-get-error-str hdbc))
```

```
)
```

Program output of the above example:

Output of (db-get-error-str hdbc)

```
ORACLE|ORA-00942|table or view does not exist
Output of (oramsg-display (db-get-error-str hdbc))

ORACLE message #0:

msg_source : ORACLE
error_code : ORA-00942
msg_string : table or view does not exist
```

db-login

Syntax

```
(db-login connection-handle data-source user-name password)
```

Description

db-login allocates the resources and performs login to a database system.

This function requires an encrypted password. If you have specified a password in the Database Setup section of the e*Way Editor, it has already been encrypted. (See **"Database Setup" on page 41**.)

If you define the password within a monk function (which is not encrypted), you must use the monk function **encrypt-password** found in the e*Gate Monk extension library stc_monkext.dll:

```
encrypt-password  encryption key   plain password
```

where encryption key is public knowledge, i.e., in this case user id, and plain password is the password to be encrypted.

The standard encrypt-password function returns an encrypted password string to be used with db-login.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
data-source	string	A data source name.
user-name	string	The database user login name.
password	string	The user login password.

Note: *The data_source, user_name, and password must not be empty strings.*

Return Values

Boolean

Returns **#t** (true) if the argument is a connection handle; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Example

```
;demo-login.monk

; define eGate path
(define EGATE "/eGate/client")

; load Monk basic extension
(define MONKLIB (string-append EGATE "/bin/stc_monkext.dll"))
(load-extension MONKLIB)
```

```
; load Monk database extension
(define STCDB "ORACLE8")
(define DARTLIB (string-append EGATE "/bin/stc_dbmonkext.dll"))
(load-extension DARTLIB)

; define data source, user ID, and password
(define dsn "Houston")
(define uid "NASA")
(define pwd (encrypt-password uid "Lunar"))

(define hdbc (make-connection-handle))
(display (string-append "\nDART Login " dsn " ...\n"))
(if (db-login hdbc dsn uid pwd)
   (begin
      (display "database login succeed !\n")
      (db-logout hdbc)
   )
   (display (db-get-error-str hdbc))
)
```

Explanation

This example shows how to use the connection handle (hbdc) to log into the data source "Houston" as "NASA" with the password "Lunar."

db-logout

Syntax

```
(db-logout connection-handle)
```

Description

db-logout performs a disconnect from the database system and releases the connection handle resources.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(define hdbc (make-connection-handle) )
(define uid "James")
(define pwd (encrypt-password uid "12345"))
(if (db-login hdbc "Payroll" "James" "12345")
    ...
)
```

Explanation

The above example shows how to disconnect from a database. For every **db-login**, there should be a corresponding **db-logout**.

Notes

Make sure to roll back or commit a transaction before you call **db-logout**. If a transaction is neither committed nor rolled back, it will be automatically rolled back before logout.

db-max-long-data-size

Syntax

```
(db-max-long-data-size connection-handle size)
```

Description

db-max-long-data specifies the maximum buffer size for the long data (SQL_LONGVARCHAR, SQL_LONGVARBINARY). Long data may have a range in size up to 2 Gigabytes ($2x10^9$). In order to limit the memory consumption of the DART library, it is necessary to use this function to specify the maximum data size expected. Long data larger than the specified size will be truncated. This data size will be used for buffer allocation for both long data columns as well as long data parameters.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
size	integer	Specifies the buffer size of the specified long data type. Note: The default buffer size is 1 megabyte.

Return Values

Boolean

Returns **#t** (true) if successful; and If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Notes

The default maximum buffer size for long data type is 1 megabyte (1048576). It is not necessary to call this function unless the long data is in excess of 1 megabyte.

db-rollback

Syntax

```
(db-rollback connection-handle)
```

Description

db-rollback rolls back the entire transaction for the connection.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if
    (and
        (db-sql-execute hdbc "delete from employee where first_name =
'John'")
        (db-sql-execute hdbc "update employee set first_name = 'Mary'
where ssn = 123456789")
    )
    (db-commit hdbc)
    (begin
        (display (db-get-error-str hdbc))
        (newline)
        (db-rollback hdbc)
    )
)
...
```

Explanation

This example shows that if the application can successfully delete “John’s record” and update “Mary’s record,” it will commit the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

db-std-timestamp-format

Syntax

```
(db-std-timestamp-format connection-handle)
```

Description

db-std-timestamp-format sets the date to SQL92 standard format—"YYYY-MM-DD HH:MI:SS.SSS"—at the connection level and must be called immediately after login.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Notes

1 When the user logs into the database, the database server will have a default timestamp format set. The default format could be any non-standard format.

Note: *The **db-std-timestamp-format** function forces the input and output of the timestamp format to the standard SQL92 standard format. Using standard format frees the user from reformatting each time data is exchanged with other applications.*

2 Oracle does not support sub-second format such as "YYYY-MM-DD HH:MI:SS." In a stored procedure a PL/SQL Table of Date data type will **ALWAYS** return in standard timestamp format.

make-connection-handle

Syntax

```
(make-connection-handle)
```

Description

make-connection-handle constructs the *connection handle*.

Parameters

None.

Return Values

A handle
Returns a connection-handle if successful, otherwise;

Boolean
Returns **#f** (false) if the function fails to create a connection-handle.Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hdbc (make-connection-handle))
```

Explanation

The above example creates a connection handle variable called hdbc.

statement-handle?

Syntax

```
(statement-handle? any-variable)
```

Description

statement-handle? determines whether or not the input argument is a statement handle data type.

Parameters

This function requires a single variable of any data type.

Return Values

Boolean

Returns **#t** (true) if the argument is a statement handle; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hstmt (db-proc-bind hdbc "test"))
(if (not (statement-handle? hstmt))
    (display (db-get-error-str hdbc))
)
```

Explanation

The above example creates a statement handle called hstmt, then it displays an error message if the newly defined hstmt is not a statement handle.

5.4 Static SQL Functions

The functions in this category control the e*Way's interaction with static SQL commands. For information about the differences between static and dynamic SQL functions, see **"Static vs. Dynamic SQL Functions" on page 119**.

The static SQL functions are:

db-sql-column-names on page 123

db-sql-column-types on page 125

db-sql-column-values on page 127

db-sql-execute on page 129

db-sql-fetch on page 130

db-sql-fetch-cancel on page 131

db-sql-format on page 132

db-sql-select on page 134

Static vs. Dynamic SQL Functions

Dynamic SQL statements are built and executed at run time versus Static SQL statements that are embedded within the program source code. Dynamic statements do not require knowledge of the complete structure of an SQL statement before building the application. This allows for run time input to provide information about the database objects to query.

The application can be written so that it prompts the user or scans a file for information that is not available at compilation time.

In Dynamic statements the four steps of processing an SQL statement take place at run time, but they are performed only once. Execution of the plan takes place only when EXECUTE is called. **Figure 34 on page 121** shows the difference between Dynamic SQL with immediate execution, and Dynamic SQL with prepared execution.

Benefits of Dynamic SQL

Using dynamic SQL commands, an application can prepare a "generic" SQL statement once and execute it multiple time. Statements can also contain markers for parameter values to be supplied at execution time, so that the statement can be executed with varying inputs.

Limitations of Dynamic SQL

The use of dynamic SQL commands has some significant limitations. A dynamic SQL implementation of an application generally performs worse than an implementation where permanent stored procedures are created and the client program invokes them with RPC (remote procedure call) commands.

Figure 33 Dynamic Statement Function Flow Chart

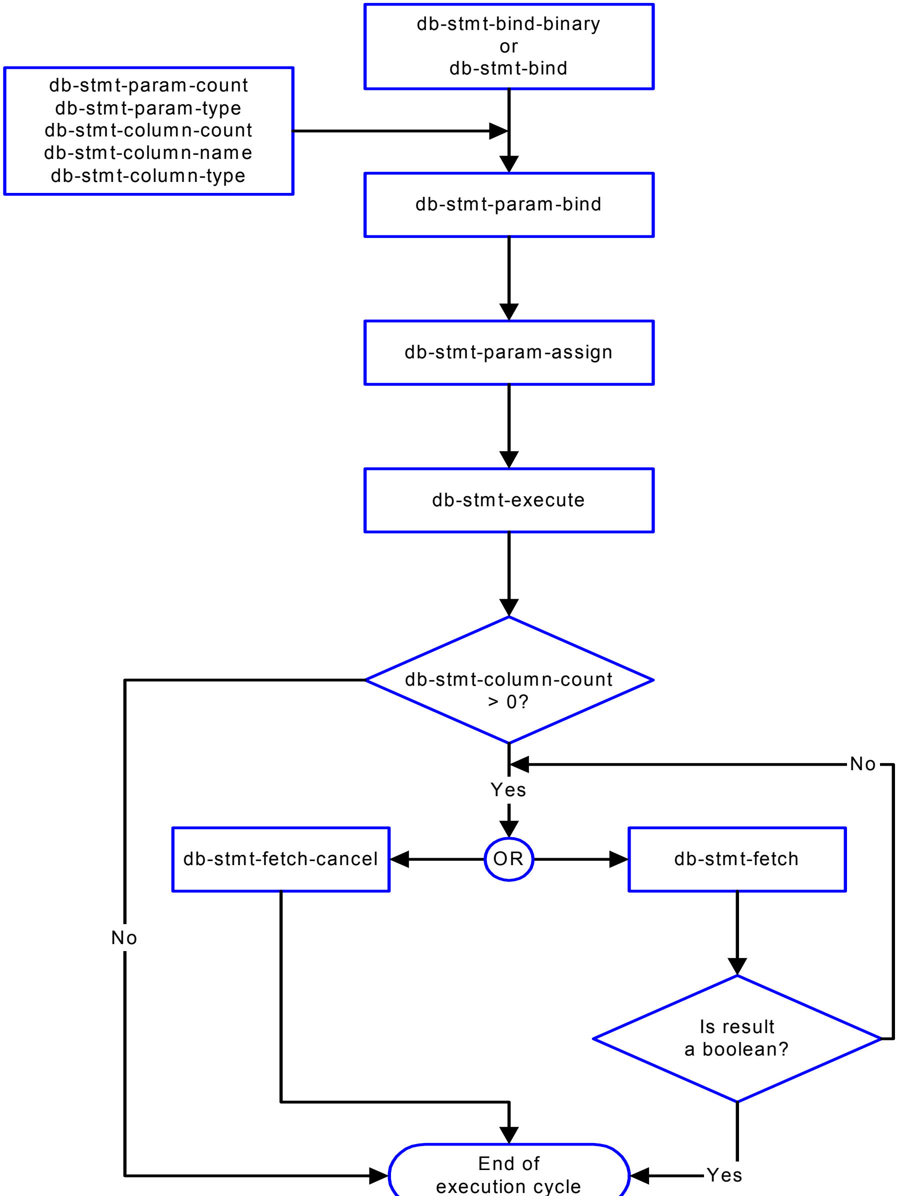
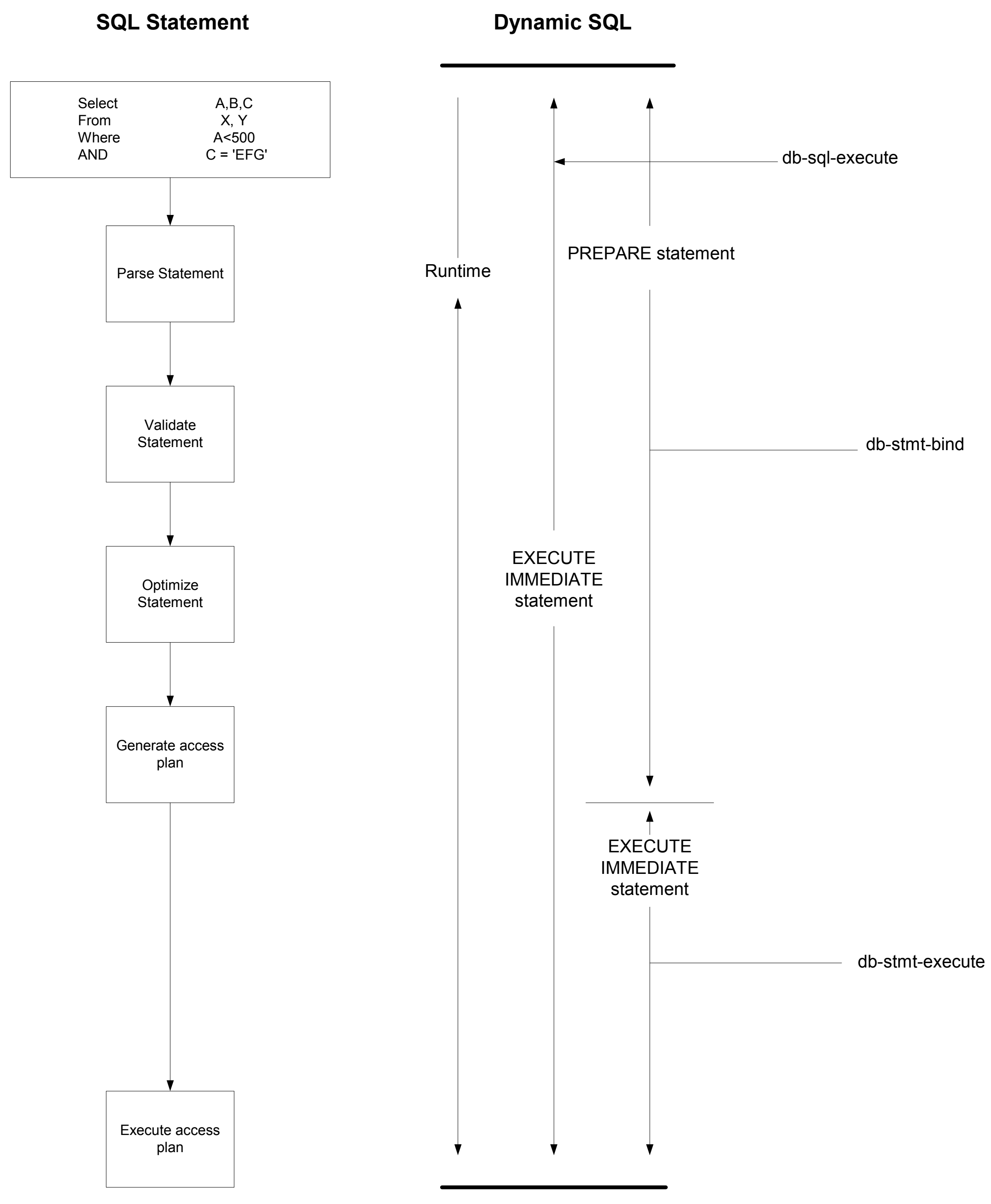


Figure 34 Example of Dynamic SQL Processing



Oracle SQL Type Support

The following table shows the supported SQL data types and the corresponding native data type for an Oracle database.

Table 9 Oracle SQL Type Support

SQL Type Name	SQL Datatype	Oracle Datatype
SQL_BIT	BIT	N/A
SQL_BINARY	BINARY (n)	N/A
SQL_VARBINARY	VARBINARY (n)	RAW (n)
SQL_CHAR	CHAR (n)	CHAR (n)
SQL_VARCHAR	VARCHAR (n)	VARCHAR2 (n)
SQL_DECIMAL	DECIMAL (p, s)	NUMBER (p, s)
SQL_NUMERIC	NUMERIC (p, s)	N/A
SQL_TINYINT	TINYINT	+
SQL_BIGINT	BIGINT	+
SQL_SMALLINT	SMALLINT	+
SQL_INTEGER	INTEGER	+
SQL_REAL	REAL	*
SQL_FLOAT	FLOAT(p)	FLOAT(b)
SQL_DOUBLE	DOUBLE PRECISION	FLOAT
SQL_DATE	DATE	N/A
SQL_TIME	TIME	N/A
SQL_TIMESTAMP	TIMESTAMP	DATE
SQL_LONGVARCHAR	LONG VARCHAR	LONG
SQL_LONGVARBINARY	LONGVARBINARY	LONG RAW

+Oracle uses number (p) to define data types that span TINYINT, BIGINT, SMALLINT, and INTEGER. Oracle integer type is internally mapped to NUMBER (38) which will be returned as SQL_INTERGER.

*Oracle float (b) specifies a floating point number with binary precision range from 1 to 126.

Note: *All variable precision data types require precision values.*

SQL_DECIMAL and SQL_NUMERIC data types require specification of scale which indicates the number of digits to the right of the decimal point.

db-sql-column-names

Syntax

```
(db-sql-column-names connection-handle selection-name)
```

Description

db-sql-column-names returns a vector of column names which are the result of an SQL SELECT statement identified by the parameter selection-name. This procedure can be called after an SQL SELECT statement has been issued successfully.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

A string

This function returns a vector of column names in string format if successful.

Boolean

If the selection-name string is unavailable for any reason, this function returns a **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title='manager'")
(if (db-login hdbc "dsn" "uid" "pwd")
    (begin
        (if (db-sql-select hdbc "manager" selection)
            (begin
                (define name-array (db-sql-column-names hdbc
"manager"))
                (if (vector? name-array)
                    (begin
                        (display "name of the first column: ")
                        (display (vector-ref name-array 0))
                        (newline)
                        ...
                    )
                    (begin
                        (display (db-get-error-str hdbc))
                        (newline)
                    )
                )
                (display (db-get-error-str hdbc))
                (newline)
            )
        )
        (if (db-alive hdbc)
            (begin
```

```
                    ...
                )
            )
            (db-logout hdbc)
        )
    )
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program will display the name of the first column.

db-sql-column-types

Syntax

```
(db-sql-column-types connection-handle selection-name)
```

Description

db-sql-column-types returns a vector of column types which are the result of an SQL SELECT statement identified by the parameter selection-name. This procedure can be called after an SQL SELECT statement has been issued successfully.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

A string

This function returns a vector of column types in string format if successful.

Boolean

If the string type is unavailable for any reason, this function returns a **#f**. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title= 'manager'")
        (if (db-sql-select hdbc "manager" selection)
            (begin
                (define type-array (db-sql-column-types hdbc
"manager"))
                (if (vector? type-array)
                    (begin
                        (display "type of the first column:")
                        (display (vector-ref type-array 0))
                        (newline)
                        ...
                        ...
                    )
                    (begin
                        (display (db-get-error-str hdbc))
                        (newline)
                    )
                )
                (display (db-get-error-str hdbc))
            )
        )
        (if (db-alive hdbc)
            (begin
                ...
            )
```

```
)
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program will display the first column type.

db-sql-column-values

Syntax

```
(db-sql-column-values connection-handle selection-name)
```

Description

db-sql-column-values returns a vector of column values, which is the result of an SQL FETCH statement identified by the parameter selection-name. This procedure can be called after an SQL FETCH statement has been issued successfully.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

A string
Returns a vector of SQL values in string format if successful.

Boolean
If the values string is unavailable for any reason, this function returns a **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title= ‘manager’")
          (if (db-sql-select hdbc "manager" selection)
               (do ((result "") (value-array #())) ((boolean? result))
                    (set! result (db-sql-fetch hdbc "manager"))
                    (if (not (boolean? reslt))
                         (begin
                              (set! value-array (db-sql-column-values hdbc
"manager"))
                          (do ((index 0 (+ index 1)) (count (vector-length
value-array))
                                   ((= index count))
                                   (display (vector-ref value-array index))
                                   (display "\t")
                              )
                              (newline)
                         )
                       (if (not result) (display (db-get-error-str hdbc)))
                    )
               )
               (begin
                    (display (db-get-error-str hdbc))
                    (newline)
               )
          )
          (if (db-alive hdbc)
```

```
          (begin
              ...
          )
      )
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program will loop through a fetch cycle. Within each fetch loop, the program displays the value of each column in the same line, separated by a tab character.

Notes

1. A successful **db-sql-fetch** call returns a string which contains the concatenation of all column values with the comma (,) character as the separator. Although this single string is suitable for display purposes, the user must parse the result string to retrieve the value of each column.

2. If the value of the column contains the comma (,) character, the user will be unable to differentiate the comma data from the comma separator. Therefore, **db-sql-column-values** returns the result as a vector of values in string type to allow the user to make use of the vector-ref function to retrieve the value of each column and avoid any parsing problem.

db-sql-execute

Syntax

```
(db-sql-execute connection-handle SQL-stmt)
```

Description

db-sql-execute executes the specified SQL statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
SQL-stmt	string	The SQL statement being executed.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc "Payroll" "James" "12345")
   (begin
        ...
        (if (db-sql-execute hdbc "insert into employee
values('John'...)")
            (db-commit hdbc)
        )
     )
     (display (db-get-error-str hdbc))
)
...
```

Explanation

This example shows that if the application can successfully log into the data source "Payroll," it will insert a record into the table "employee."

Notes

1. Use the **db-sql-select** function to execute a select statement.
2. Use **db-commit** or **db-rollback** to commit and roll back transactions.

db-sql-fetch

Syntax

```
(db-sql-fetch connection-handle selection-name)
```

Description

db-sql-fetch “fetches” the result of a SELECT statement. The statement handle is “free” after the function fetches the last record.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

A string
Returns a comma, delimited string containing all the column values for the record.

Boolean
Returns **#t** (true) at the end of the “fetch cycle,” when no more records are available to “fetch"; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-sql-select hdbc "GreaterThan25" "select * employee where age
> 25")
    (begin
        (display (db-sql-fetch hdbc "GreaterThan25"))
        (newline)
        (db-sql-fetch-cancel hdbc "GreaterThan25")
    )
    (begin
        (display (db-get-error-str hdbc))
        (newline)
    )
)
...
```

Explanation

This example shows that the application selects the first record of employees who are older than age 25, by fetching the record once and cancelling the rest of the records.

Notes

The return result is temporarily stored in RAM. The buffer is allocated when **db-sql-select** is called. The maximum size of the buffer is determined by the operating system.

db-sql-fetch-cancel

Syntax

```
(db-sql-fetch-cancel connection-handle selection-name)
```

Description

db-sql-fetch-cancel closes the cursor associated with an SQL SELECT statement and cancels the fetch command. It also frees up the memory allocation for the selection.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-sql-select hdbc "GreaterThan25" "select * employee where age
> 25")
    (begin
        (define result (db-sql-fetch hdbc "GreaterThan25"))
        (if (not (boolean? result))
            (db-sql-fetch-cancel hdbc "GreaterThan25")
            (if (not result)
                (begin
                    (display (db-get-error-str hdbc))
                    (newline)
                )
            )
        )
    )
    (begin
        (display (db-get-error-str hdbc))
        (newline)
    )
)
...
```

Explanation

This example shows that the application selects the first record of employees who are older than age 25, by fetching the record once and cancelling the rest of the records.

db-sql-format

Syntax

```
(db-sql-format data-string SQL-type)
```

Description

db-sql-format returns a formatted string of the *data-string*, so it can be used in an SQL statement as a literal value of a corresponding *SQL-type*.

In the current implementation, only the SQL_CHAR, SQL_VARCHAR, SQL_DATE, SQL_TIME, and SQL_TIMESTAMP SQL-types will be formatted. If the *data-string* is an empty string, the procedure will return a NULL value for all SQL data types except SQL_CHAR and SQL_VARCHAR.

Parameters

Name	Type	Description
data-string	string	A data string to be used as a literal value in an SQL statement.
SQL-type	string	An SQL datatype string, i.e., SQL_VARCHAR.

Return Values

A string

Returns a formatted string used as a literal data value in an SQL statement.

Throws

None.

Examples

```
(define last-name (db-sql-format "O'Reilly" "SQL_VARCHAR"))
(define timestamp (db-sql-format "1998-02-19 12:34:56"
SQL_TIMESTAMP"))
(define sql-stmt (string-append "update employee set lastname =
"last-name ", MODIFYTIME = "timestamp "WHERE SSN = 123456789"))
(if (db-login hdbc "Payroll" "user" "password")
    (begin
        (if (db-sql-execute hdbc sql-stmt)
            (db-commit hdbc)
            (begin
                (display (db-get-error-str hdbc))
                (newline)
                (db-rollback hdbc)
            )
        )
        ...
        (db-logout hdbc)
    )
)
```

Explanation

The example above illustrates how the program uses **db-sql-format** to format the last name and the timestamp and use the results as part of an SQL statement.

Notes

1. This function will only work with the **db-std-timestamp-format** function since the **db-sql-format** function handles only standard timestamp format.
2. The (timestamp) Monk built-in function is used to insert the timestamp to an Event Type Definition. You should specify the following format for it to be accepted by the **db-sql-format** function:

```
%Y-%m-%d  %H:%M:%S
```

For SQL_CHAR and SQL_VARCHAR (SQL data types) **db-sql-format** will place a single quotation mark (‘) before and after the *data-string*, and expand each single quotation mark in the *data-string* to two single quotation mark characters.

The following table shows the typical *data-string* and the corresponding result of the formatting for these SQL types.

Table 10 SQL Statement Format

SQL_type Value	Data_string Value	Formatted Result String
SQL_CHAR	This is a string	‘This is a string.’
SQL_VARCHAR	O’Reilly	‘O’ ‘Reilly’
SQL_TIMESTAMP	1998-02-19 12:34:56.789	‘1998-02-19 12:34:56’

db-sql-select

Syntax

```
(db-sql-select connection-handle selection-name SQL-statement)
```

Description

db-sql-select executes an SQL SELECT statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The selection identifier.
SQL-statement	string	The SELECT statement used.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-sql-select hdbc "GreaterThan25" "select * employee where age
> 25")
    (begin
        (display (db-sql-fetch hdbc "GreaterThan25"))
        (newline)
        (db-sql-fetch-cancel hdbc "GreaterThan25")
    )
    (display (db-get-error-str hdbc))
)
...
```

Explanation

This example shows that the application selects the first record of employees who are older than age 25 by fetching the records one at a time and cancelling the remainder of the return records.

5.5 Dynamic SQL Functions

The functions in this category control the e*Way's interaction with dynamic SQL commands. For information about the differences between static and dynamic SQL functions, see **"Static vs. Dynamic SQL Functions" on page 119**.

The dynamic SQL functions are:

db-stmt-bind on page 136

db-stmt-bind-binary on page 137

db-stmt-column-count on page 138

db-stmt-column-name on page 139

db-stmt-column-type on page 140

db-stmt-execute on page 141

db-stmt-fetch on page 142

db-stmt-fetch-cancel on page 143

db-stmt-param-assign on page 144

db-stmt-param-bind on page 145

db-stmt-param-count on page 146

db-stmt-param-type on page 147

db-stmt-row-count on page 148

db-stmt-bind

Syntax

```
(db-stmt-bind connection-handle dynamic-SQL-statement)
```

Description

db-stmt-bind binds the dynamic statement specified. The binary data type should be input or output parameters with hexadecimal format.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
dynamic-SQL-statement	string	The dynamic statement to be bound

Return Values

Statement handle
The statement handle that identifies the dynamic statement specified.

Boolean
If unsuccessful, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Additional Information

If the user needs to input /output binary data in the raw (binary) format, they should use **db-stmt-bind-binary**.

Notes

1. Oracle OCI API is not able to report the data type for each bound parameter in a dynamic statement. All bound parameters will default to VARCHAR data types. This will allow Oracle to implicitly convert the data string of each parameter into the correct data value of the parameter at the execution of the dynamic statement.
2. If the user needs to select long data type column, the long column should appear at the end of the selection list.

Example

```
...
(db-stmt-bind connection-handle "select last_name from db_employee
where emp_no = 155")
...
```

db-stmt-bind-binary

Syntax

```
(db-stmt-bind-binary connection-handle dynamic-SQL-statement)
```

Description

db-stmt-bind-binary binds the dynamic statement specified. The binary data type will be input and output with raw format.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
dynamic-SQL-statement	string	The dynamic statement to be bound

Return Values

Statement handle
The statement handle that identifies the dynamic statement specified.

Boolean
If unsuccessful, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-column-count

Syntax

```
(db-stmt-column-count connection-handle statement-handle)
```

Description

db-stmt-column-count returns the number of columns in the return result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

A number
Returns a number greater than zero (0) when the record set is available.

Boolean
If no record set is available, the return value will be **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-column-name

Syntax

```
(db-stmt-column-name connection-handle statement-handle index)
```

Description

db-stmt-column-name returns the name string of the specified column in the result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.
index	integer	An integer equal to --0 to db-stmt-column-count minus 1.

Return Values

A string
Returns the name string if successful.

Boolean
If unsuccessful, returns **#f** (false). Use db-get-error-str to retrieve the error message.

Throws

None.

db-stmt-column-type

Syntax

```
(db-stmt-column-type connection-handle statement-handle index)
```

Description

db-stmt-column-type returns the SQL data type of the specified column in the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.
index	integer	An integer equal to --0 to db-stmt-column-count minus 1.

Return Values

A string
Returns a string of SQL data type when successful.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-execute

Syntax

```
(db-stmt-execute connection-handle statement-handle)
```

Description

db-stmt-execute executes the dynamic statement of a specified statement-handle.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-stmt-bind.

Return Values

Boolean

If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-fetch

Syntax

```
(db-stmt-fetch connection-handle statement-handle)
```

Description

db-stmt-fetch retrieves the column values of the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

A Vector and a Boolean
Returns a vector containing all the column values and at the end of the "fetch cycle" returns **#t** (true) when no more records are available to "fetch."

Boolean
If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-fetch-cancel

Syntax

```
(db-stmt-fetch-cancel connection-handle statement-handle)
```

Description

db-stmt-fetch-cancel terminates the current “fetch” cycle.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement-handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Notes

The **db-stmt-fetch-cancel** function (and other **db-stmt-xxxx** functions) will never close its associated cursor, because it is designed to bind once and execute multiple times. The same dynamic statement can be executed multiple times without the need to reopen the cursor and rebind the same statement. This conserves processing time by reducing the amount of parsing. However, it is important to know that the cursor will not be closed until the function disconnects from the database.

db-stmt-param-assign

Syntax

```
(db-stmt-param-assign connection-handle statement-handle index value)
```

Description

db-stmt-param-assign assigns the parameter and executes the dynamic statement of a specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-proc-bind.
index	integer	The number between 0 and db-stmt-param-count minus 1.
value	string	The value to be assigned to the parameter.

Return Values

Boolean

If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-param-bind

Syntax

```
(db-stmt-bind hdbc hstmt index sqltype precision scale)
```

Description

db-stmt-param-bind binds the each of the input parameters properties of the dynamic statement specified.

Parameters

Name	Type	Description
hdbc	connection handle	A connection handle to the database
hstmt	string	The statement handle that identifies the stored procedure specified.
index	integer	The number between 0 and db-stmt-param-count-minus 1.
sqltype	string	The string that identifies the SQL type being used.
precision	integer, including decimal points	The number of places to the right and left of the decimal point to represent the total amount of space occupied by the SQL type.
scale	integer	The number of places to the right of the decimal point.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

db-stmt-param-count

Syntax

```
(db-stmt-param-count connection-handle  statement-handle
```

Description

db-stmt-param-count retrieves the number of parameters in the dynamic statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

An Integer

Returns a number, which represents the number of parameters for the dynamic statement specified, when successful.

Boolean

If unsuccessful, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-param-type

Syntax

```
(db-stmt-param-type connection-handle statement-handle index)
```

Description

db-stmt-param-type retrieves the SQL data type of the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the stored procedure specified. This is the handle produced by db-stmt-bind.
index	integer	The number between 0 and db-stmt-param-count minus 1.

Return Values

A string

If successful, **db-stmt-param-type** returns a string which represents the SQL data type.

Boolean

If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-row-count

Syntax

```
(db-stmt-row-count connection-handle statement-handle index)
```

Description

db-stmt-column-size returns the number of rows affected by the execution of the SQL statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.
index	integer	An integer equal to --0 to db-stmt-column-count minus 1.

Return Values

Boolean

If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

5.6 Stored Procedure Functions

The functions in this category control the e*Way's interaction with stored procedures.

The stored procedure functions are:

db-proc-bind on page 151

db-proc-bind-binary on page 152

db-proc-column-count on page 153

db-proc-column-name on page 155

db-proc-column-type on page 157

db-proc-execute on page 159

db-proc-fetch on page 161

db-proc-fetch-cancel on page 163

db-proc-max-records on page 164

db-proc-param-assign on page 165

db-proc-param-count on page 167

db-proc-param-io on page 169

db-proc-param-name on page 170

db-proc-param-type on page 171

db-proc-param-value on page 172

db-proc-return-exist on page 174

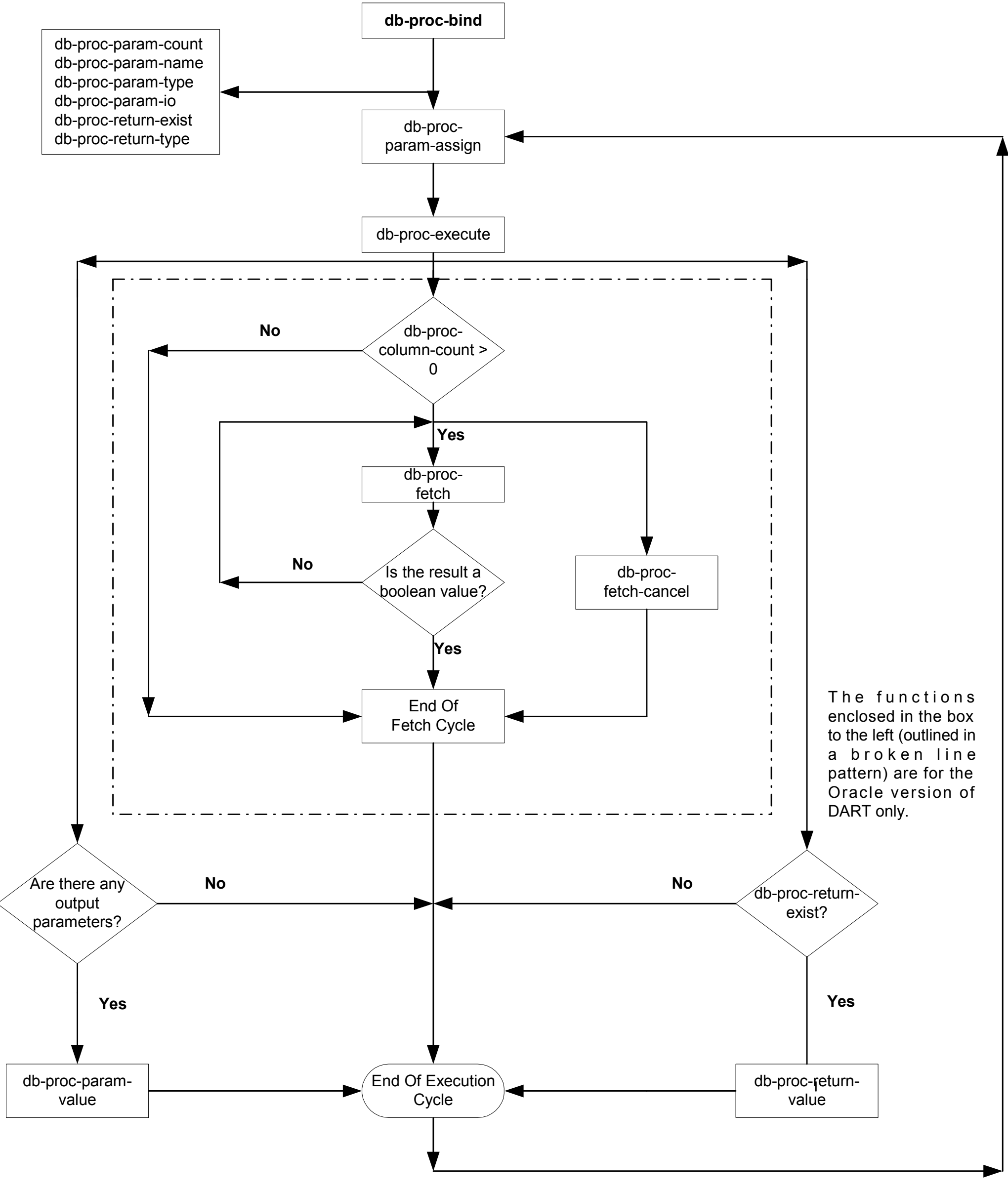
db-proc-return-type on page 176

db-proc-return-value on page 178

Benefits of Stored Procedures

When a stored procedure is created for an application, SQL statement compilation and optimization are performed once when the procedure is created. With a dynamic SQL application, compilation and optimization are performed every time the client program runs. A dynamic SQL implementation also incurs database space overhead because each instance of the client program must create separate compiled versions of the application's prepared statements. When you design an application to use stored procedures and RPC commands, all instances of the client program can share the same stored procedures. (See **Figure 35 on page 150**)

Figure 35 Calling a Stored Procedure (Oracle)



db-proc-bind

Syntax

```
(db-proc-bind connection-handle procedure-name)
```

Description

db-proc-bind binds the input/output parameters of the stored procedure specified.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
procedure-name	string	The stored procedure to be bound.

Return Values

A string
Returns a statement-handle when successful; otherwise,

Boolean
Returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hstmt (db-proc-bind hdbc "test")
(if (not (statement-handle? hstmt)
    (display "fail to bind stored procedure test\n")
)
```

db-proc-bind-binary

Syntax

```
(db-proc-bind-binary connection-handle dynamic-SQL-statement)
```

Description

db-proc-bind-binary binds the dynamic statement specified. The format of the input and output data is binary.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
dynamic-SQL-statement	string	The dynamic statement to be bound

Return Values

A string
Returns a statement-handle when successful; otherwise,

Boolean
Returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-proc-column-count

Syntax

```
(db-proc-column-count connection-handle statement-handle)
```

Description

db-proc-column-count retrieves the number of columns in the return result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

A number
Returns a number greater than zero (0) when the record set is available.

Boolean
If no record set is available, the return value will be **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
       (if (db-proc-param-assign hdbc hstmt 0 "5")
         (if (db-proc-execute hdbc hstmt)
           (begin
              (define col-count (db-proc-column-count))
              (if (and (number? col-count) (> col-count 0))
                (do ((i 0 (+ i 1))) ((= i col-count))
                   (display "column ")
                   (display (db-proc-column-name hdbc hstmt i))
                   (display ": type = ")
                   (display (db-proc-column-type hdbc hstmt i))
                   (newline)
                )
                (display (db-get-error-str hdbc))
              )
              ...
              ...
           )
           (display (db-get-error-str hdbc))
       )
       (display (db-get-error-str hdbc))
```

```
            )
            (display (db-get-error-str hdbc))
        )
        ...
        ...
        )
          (display (db-get-error-str hdbc)
      )
```

Notes

Oracle does not provide a simple mechanism for returning multiple records from the stored procedure. The PL/SQL table type is used to contain the multiple records to be returned. After binding the stored procedure, **db-proc-column-count** returns the number of PL/SQL table types that contain multiple return records.

The Oracle procedure to return result set is given here:

```
oracle_odbc.sql
```

db-proc-column-name

Syntax

```
(db-proc-column-name connection-handle statement-handle column-index)
```

Description

db-proc-column-name retrieves the name string of the specified column in the result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
column-index	string	The SQL data type of the specified column in the results set--0 to db-proc-column-count minus 1.

Return Values

A string

Returns the name string if successful.

Boolean

If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
       (if (db-proc-param-assign hdbc hstmt 0 "5")
         (if (db-proc-execute hdbc hstmt)
           (begin
              (define col-count (db-proc-column-count))
              (if (and (number? col-count) (> col-count 0))
                (do ((i 0 (+ i 1))) ((= i col-count))
                   (display "column ")
                   (display (db-proc-column-name hdbc hstmt i))
                   (display ": type = ")
                   (display (db-proc-column-type hdbc hstmt i))
                   (newline)
                )
                (display (db-get-error-str hdbc))
              )
              ...
              ...
```

```
                    )
                    (display (db-get-error-str hdbc))
                  )
                  (display (db-get-error-str hdbc))
              )
              (display (db-get-error-str hdbc))
          )
              ...
              ...
          )
           (display (db-get-error-str hdbc)
      )
```

Notes

Since the result set of a stored procedure is returned through the parameters of the PL/SQL table type, the name of the table type parameter will be returned.

db-proc-column-type

Syntax

```
(db-proc-column-type connection-handle statement-handle column-index)
```

Description

db-proc-column-type retrieves the SQL data type of the specified column in the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
column-index	string	The SQL data type of the specified column in the record set--0 to db-proc-column-count minus 1.

Return Values

A string

Returns a string of SQL data type when successful.

Boolean

If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
       (if (db-proc-param-assign hdbc hstmt 0 "5")
         (if (db-proc-execute hdbc hstmt)
           (begin
              (define col-count (db-proc-column-count))
              (if (and (number? col-count) (> col-count 0))
                (do ((i 0 (+ i 1))) ((= i col-count))
                   (display "column ")
                   (display (db-proc-column-name hdbc hstmt i))
                   (display ": type = ")
                   (display (db-proc-column-type hdbc hstmt i))
                   (newline)
                )
                (display (db-get-error-str hdbc))
              )
              ...
              ...
```

```
                  )
                  (display (db-get-error-str hdbc))
                )
                (display (db-get-error-str hdbc))
              )
              (display (db-get-error-str hdbc))
            )
            ...
            ...
         )
          (display (db-get-error-str hdbc)
       )
```

Notes

Since the result set of the stored procedure is returned through the parameters of the PL/SQL table type, a PL/SQL table can only contain one standard Oracle data type.

db-proc-execute

Syntax

```
(db-proc-execute connection-handle statement-handle)
```

Description

db-proc-execute executes a stored procedure.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
       (if (db-proc-param-assign hdbc hstmt 0 "5")
         (if (db-proc-execute hdbc hstmt)
             ...
             (display (db-get-error-str hdbc))
         )
         (display (db-get-error-str hdbc))
      )
       (display (db-get-error-str hdbc))
     )
     ...
     ...
     (db-logout hdbc)
  )
  (display (db-get-error-str hdbc))
)
```

Notes

The default precision for number or real type is 38 for a column in the table. This is important when executing a stored procedure that retrieves values from that column in the table. The **db-proc-execute** function will fail if the exponential part of the value is larger than 38.

For example:

- 1.555E+38 is acceptable
- 1.55E+39 will prevent the successful retrieval of the column values

db-proc-fetch

Syntax

```
(db-proc-fetch connection-handle statement-handle)
```

Description

db-proc-fetch retrieves the column values of the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

A vector and Boolean

Returns a vector containing all the column values and at the end of the "fetch cycle" returns **#t** (true) when no more records are available to "fetch."

Boolean

If unsuccessful, this function returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
   (begin
      (display "database login succeed !\n")
      (define hstmt (db-proc-bind hdbc "TEST_PROC"))
      (if (statement-handle? hstmt)
         (if (db-proc-param-assign hdbc hstmt 0 "5")
            (if (db-proc-execute hdbc hstmt)
               (begin
                  (define col-count (db-proc-column-count))
                  (if (and (number? col-count) (> col-count 0))
                    (do ((result (db-proc-fetch hdbc hstmt) (db-proc-
fetch hdbc hstmt)))
                       ((boolean? result) (begin (display result)
(newline)))
                       (display result
                       (newline)
                    )
                  )
                  ...
                  ...
               )
                            (display (db-get-error-str hdbc)
            )
                    (display (db-get-error-str hdbc)
```

```
                )
                            (display (db-get-error-str hdbc)
            )
            ...
            ...
        )
        (display (db-get-error-str hdbc)
    )
```

db-proc-fetch-cancel

Syntax

```
(db-proc-fetch-cancel connection-handle statement-handle)
```

Description

db-proc-fetch-cancel terminates the current “fetch” cycle.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
   (begin
      (display "database login succeed !\n")
      (define hstmt (db-proc-bind hdbc "TEST_PROC"))
      (if (statement-handle? hstmt)
          (if (db-proc-param-assign hdbc hstmt 0 "5")
            (if (db-proc-execute hdbc hstmt)
               (begin
                  (define col-count (db-proc-column-count))
                  (if (and (number? col-count) (> col-count 0))
                    (db-proc-fetch-cancel hdbc hstmt)
               )
            )
            ...
            ...
          )
                   (display (db-get-error-str hdbc)
      )
            (display (db-get-error-str hdbc)
     )
          (display (db-get-error-str hdbc)
      )
   ...
   ...
   )
   (display (db-get-error-str hdbc)
)
```

db-proc-max-records

Syntax

```
(db-proc-max-records connection-handle num_of_records)
```

Description

db-proc-max-records sets the maximum number of records that the Oracle e*Way can hold for a stored procedure. By default, the maximum number of records is set to 100.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
num_of_records	integer	The number of records the Oracle e*Way can hold for a stored procedure.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
    (begin
        (display "database login succeed !\n")
        (if (db-proc-max-records hdbc 1000)
         ...
        )
        ...
    )
    ...
)
```

Notes

1. This function should be called as soon as the connection has been established and before binding the stored procedure. At most, it should be called only once for each connection.
2. The number of records that can be returned by the Oracle e*Way for a stored procedure is dependent upon the memory available and should not be greater than 32512 (which is the maximum number of records that can be returned by an Oracle stored procedure). Keep in mind that large numbers of records or large records use extensive resources and require more processing time.

db-proc-param-assign

Syntax

```
(db-proc-param-assign connection-handle statement-handle param-
index param-value)
```

Description

db-proc-param-assign "assigns" the value of an IN or INOUT parameter and places that value into internal storage.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.
param-value	string	The input value of the IN or INOUT parameter.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

Scenario #1 - sample code for db-proc-param-assign

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
       (if (db-proc-param-assign hdbc hstmt 0 "5")
         (if (db-proc-execute hdbc hstmt)
             ...
             (display (db-get-error-str hdbc))
         )
         (display (db-get-error-str hdbc))
      )
       (display (db-get-error-str hdbc))
     )
     ...
     ...
     (db-logout hdbc)
  )
  (display (db-get-error-str hdbc))
```

Scenario #2 — sample code for db-proc-param-assign with multiple input arguments

```
(if (db-login hdbc dsn uid pwd)
  (begin
    (display "database login succeed !\n")
    (define hstmt (db-proc-bind hdbc "TEST_PROC"))
    (if (statement-handle? hstmt)
        (if (and
              (db-proc-param-assign hdbc hstmt 0 "5")
              (db-proc-param-assign hdbc hstmt 2 "O'REILLY")
              (db-proc-param-assign hdbc hstmt 7 "1A2B78F0")
            )
            (if (db-proc-execute hdbc hstmt)
             ...
             ...
            )
            (display (db-get-error-str hdbc))
        )
        (display (db-get-error-str hdbc))
    )
    ...
    ...
  )
  (display (db-get-error-str hdbc))
```

Notes

1. The value for the param-value argument should be entered as a string, without enclosure in single quotation marks (‘) for SQL_CHAR and SQL_VARCHAR.
2. The literal value for SQL_BINARY and SQL_VARBINARY should be a hexadecimal string. Refer to Scenario #2 above.

db-proc-param-count

Syntax

```
(db-proc-param-count connection-handle statement-handle)
```

Description

db-proc-param-count retrieves the number of parameters in the stored procedure.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

A integer
Returns a integer, which represents the number of parameters for the stored procedure specified, when successful.

Boolean
If the number is unavailable due to a problem within one of the arguments, the function returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
    (begin
        (display "database login succeed !\n")
        (define hstmt (db-proc-bind hdbc "TEST_PROC"))
        (if (statement-handle? hstmt)
          (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
              (display "parameter ")
               (display (db-proc-param-name hdbc hstmt i))
              (display ": type = ")
              (display (db-proc-param-type hdbc hstmt i))
              (display ", io = ")
              (display (db-proc-param-io hdbc hstmt i))
              (newline)
           )
           (display (db-get-error-str hdbc))
        )
        ...
        ...
    )
    (display (db-get-error-str hdbc))
)
```

Notes

The PL/SQL table type parameter is treated as a column rather than a parameter because it contains multiple values. A parameter contains only one value. Because of this the return value of this function will be the number of non-table type parameters only. The **db-proc-column-count** function will return the number of table type parameters.

db-proc-param-io

Syntax

```
(db-proc-param-io connection-handle statement-handle param-index)
```

Description

db-proc-param-io retrieves the IO type for the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

A string
Returns an IO type string as **IN**, **OUT**, or **INOUT**

Boolean
otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
    (begin
        (display "database login succeed !\n")
        (define hstmt (db-proc-bind hdbc "TEST_PROC"))
        (if (statement-handle? hstmt)
          (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
              (display "parameter ")
              (display (db-proc-param-name hdbc hstmt i))
              (display ": type = ")
              (display (db-proc-param-type hdbc hstmt i))
              (display ", io = ")
              (display (db-proc-param-io hdbc hstmt i))
              (newline)
           )
           (display (db-get-error-str hdbc))
         )
        ...
        ...
    )
    (display (db-get-error-str hdbc))
)
```

db-proc-param-name

Syntax

```
(db-proc-param-name connection-handle statement-handle param-index)
```

Description

db-proc-param-name retrieves the name of the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

A string
Returns the string containing the name of the parameter.

Boolean
Returns **#f** (false) if unable to return the string containing the name of the parameter. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
    (begin
        (display "database login succeed !\n")
        (define hstmt (db-proc-bind hdbc "TEST_PROC"))
        (if (statement-handle? hstmt)
          (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
              (display "parameter ")
              (display (db-proc-param-name hdbc hstmt i))
              (display ": type = ")
              (display (db-proc-param-type hdbc hstmt i))
              (display ", io = ")
              (display (db-proc-param-io hdbc hstmt i))
              (newline)
           )
           (display (db-get-error-str hdbc))
         )
        ...
        ...
    )
    (display (db-get-error-str hdbc))
)
```

db-proc-param-type

Syntax

```
(db-proc-param-type connection-handle statement-handle param-index)
```

Description

db-proc-param-type retrieves the SQL data type of the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

A string

If successful, **db-proc-param-type** returns a string which represents the SQL data type.

Boolean

If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
    (begin
        (display "database login succeed !\n")
        (define hstmt (db-proc-bind hdbc "TEST_PROC"))
        (if (statement-handle? hstmt)
          (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
              (display "parameter ")
              (display (db-proc-param-name hdbc hstmt i))
              (display ": type = ")
              (display (db-proc-param-type hdbc hstmt i))
              (display ", io = ")
              (display (db-proc-param-io hdbc hstmt i))
              (newline)
           )
           (display (db-get-error-str hdbc))
         )
        ...
        ...
    )
    (display (db-get-error-str hdbc))
)
```

db-proc-param-value

Syntax

```
(db-proc-param-value connection-handle statement-handle param-
index)
```

Description

db-proc-param-value retrieves the value of the **OUT** or **INOUT** parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

A string
Returns a string which represents the value of the **OUT** or **INOUT** parameter.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
    (begin
      (display "database login succeed !\n")
      (define hstmt (db-proc-bind hdbc "TEST_PROC"))
      (if (statement-handle? hstmt)
          (if (db-proc-param-assign hdbc hstmt 0 "5")
              (if (db-proc-execute hdbc hstmt)
                  (begin
                      (define col-count (db-proc-column-count hdbc hstmt))
                      (if (and (number? col-count) (> col-count 0))
                          (do ((result (db-proc-fetch hdbc hstmt) (db-proc-
fetch hdbc hstmt)))
                              ((boolean? result))
                              (display result)
                              (newline)
                          )
                      )
                      (define prm-count (db-proc-param-count hdbc hstmt))
                      (do ((i 0 (+ i 1))) ((= i prm-count))
                          (if (not (equal? (db-proc-param-io hdbc hstmt i)
"IN"))
                              (begin
                                  (display "output parameter ")
```

```
                    (display (db-proc-param-name hdbc hstmt i))
                    (display "= ")
                   (display (db-proc-param-value hdbc hstmt i))
                    (newline)
                )
            )
        )
        ...
        ...
    )
```

Notes

The parameter value will be made available after the stored procedure has been executed.

db-proc-return-exist

Syntax

```
(db-proc-return-exist connection-handle statement-handle)
```

Description

db-proc-return-exist determines whether or not the stored procedure has a return value.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

Boolean

Returns **#t** (true) if a return value exists or **#f** (false) when no return value exists or an error occurs. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
          (if (statement-handle? hstmt)
            (if (db-proc-param-assign hdbc hstmt 0 "5")
              (if (db-proc-execute hdbc hstmt)
                (begin
                   (define col-count (db-proc-column-count))
                   (if (and (number? col-count) (> col-count 0))
                    (do ((result (db-proc-fetch hdbc hstmt) (db-proc-
fetch hdbc hstmt)))
                       ((boolean? result))
                       (display result)
                       (newline)
                     )
                   )
                   (if (db-proc-return-exist hdbc hstmt)
                     (begin
                        (display "return type = ")
                        (display (db-proc-return-type hdbc hstmt))
                        (newline)
                        (display "return value = ")
                        (display (db-proc-return-value hdbc hstmt))
                        (newline)
                     )
                   )
                   ...
```

```
                              ...
                           )
                           (display (db-get-error-str hdbc))
                         )
                         (display (db-get-error-str hdbc))
                       )
                       (display (db-get-error-str hdbc))
                     )
                ...
                ...
            )
             (display (db-get-error-str hdbc)
          )

          )
                           (display (db-get-error-str hdbc))
                         )
                         (display (db-get-error-str hdbc))
                       )
                       (display (db-get-error-str hdbc))
                     )
                ...
                ...
            )
```

db-proc-return-type

Syntax

```
(db-proc-return-type connection-handle statement-handle)
```

Description

db-proc-return-type determines the SQL data type for the return value.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

A string

Returns an SQL data type string, i.e., SQL_VARCHAR.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
          (if (statement-handle? hstmt)
            (if (db-proc-param-assign hdbc hstmt 0 "5")
              (if (db-proc-execute hdbc hstmt)
                (begin
                   (define col-count (db-proc-column-count))
                   (if (and (number? col-count) (> col-count 0))
                    (do ((result (db-proc-fetch hdbc hstmt) (db-proc-
fetch hdbc hstmt)))
                       ((boolean? result))
                       (display result)
                       (newline)
                     )
                   )
                   (if (db-proc-return-exist hdbc hstmt)
                     (begin
                        (display "return type = ")
                        (display (db-proc-return-type hdbc hstmt))
                        (newline)
                        (display "return value = ")
                        (display (db-proc-return-value hdbc hstmt))
                        (newline)
                     )
                   )
                   ...
                   ...
                )
```

```
                    (display (db-get-error-str hdbc))
                  )
                  (display (db-get-error-str hdbc))
                )
                (display (db-get-error-str hdbc))
              )
              ...
              ...
      )
       (display (db-get-error-str hdbc)
    )
```

Notes

The stored functions defined in the Oracle DBMS can have any SQL data type as the return value.

db-proc-return-value

Syntax

```
(db-proc-return-value connection-handle statement-handle)
```

Description

db-proc-return-value retrieves the return value (return status) for the stored procedure.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

A string

Returns a string which represents the return value.

Throws

None.

Examples

```
(begin
     (display "database login succeed !\n")
     (define hstmt (db-proc-bind hdbc "TEST_PROC"))
          (if (statement-handle? hstmt)
            (if (db-proc-param-assign hdbc hstmt 0 "5")
              (if (db-proc-execute hdbc hstmt)
                (begin
                   (define col-count (db-proc-column-count))
                   (if (and (number? col-count) (> col-count 0))
                    (do ((result (db-proc-fetch hdbc hstmt) (db-proc-
fetch hdbc hstmt)))
                        ((boolean? result))
                        (display result)
                        (newline)
                      )
                   )
                   (if (db-proc-return-exist hdbc hstmt)
                     (begin
                        (display "return type = ")
                        (display (db-proc-return-type hdbc hstmt))
                        (newline)
                        (display "return value = ")
                        (display (db-proc-return-value hdbc hstmt))
                        (newline)
                     )
                   )
                   ...
                   ...
                )
                (display (db-get-error-str hdbc))
```

```
                    )
                    (display (db-get-error-str hdbc))
                  )
                  (display (db-get-error-str hdbc))
                )
           ...
           ...
        )
         (display (db-get-error-str hdbc)
      )
```

Notes

1. Stored procedures can return an integer value called a return status. This status indicates that the procedure completed successfully or shows the reason for failure. SQL Server has a defined set of return values; or users can define their own return values.
2. The SQL Server reserves 0 to indicate a successful return, and negative values in the range of -1 to -99 are assigned to a listing of reasons for failure. Numbers 0 and -1 to -14 are in use currently.

Value	Meaning
0	procedure executed without error
-1	missing object
-2	datatype error
-3	process was chosen as deadlock victim
-4	permission error
-5	syntax error
-6	miscellaneous user error
-7	resource error, such as out of space
-8	non-fatal internal problem
-9	system limit was reached
-10	fatal internal inconsistency
-11	fatal internal inconsistency
-12	table or index is corrupt
-13	database is corrupt
-14	hardware error

5.7 Message Event Functions

The functions in this category control the e*Way's message Event operations.

The message Event functions are:

db-struct-bulk-insert on page 181

db-struct-call on page 182

db-struct-execute on page 183

db-struct-fetch on page 184

db-struct-insert on page 186

db-struct-select on page 188

db-struct-update on page 190

db-struct-bulk-insert

Syntax

```
(db-struct-bulk-insert connection-handle table-path)
```

Description

db-struct-bulk-insert inserts an Event Type Definition with repeating nodes (for example, multiple records) into a table.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	path	A path which represents a table.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error notification.

Throws

None.

Notes

1. The Event type MUST be a fixed-length, which can be generated using **stcstruct.exe** with the -f option.
2. The number of records that can be inserted into a table is dependent on the memory available and cannot be greater than 32512.

   The format of the literal value of the SQL_DECIMAL and SQL_TIMESTAMP data type is dependent on the national language support parameter of the SQL server. You can use the SQL statement ALTER SESSION to modify the date format and the decimal character. For example:

   ```
   alter session set NLS_DATE_FORMAT= 'DD-MON-YY'
   alter session set NLS_NUMERIC_CHARACTERS = '.,'
   ```
3. To insert a NULL value into a table, specify binary 0 in the VALUE field of a node.

   For example:

   ```
   (make-string 1 (integer->char(0)))
   ```

db-struct-call

Syntax

```
(db-struct-call connection-handle statement-handle procedure-path)
```

Description

db-struct-call calls the stored procedure using the value from the *procedure-path* node of the DART Event Type Definition, retrieves all procedure output and places this information into the DART Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection-handle	A connection handle to the database.
statement-handle	statement-handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
procedure-path	path	The absolute path to the procedure nodes in the DART Event Type Definition.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

db-struct-execute

Syntax

```
(db-struct-execute connection-handle statement-handle statement-path)
```

Description

db-struct-execute calls the dynamic statement using the value from the *statement-path* node of the Event Type Definition, retrieves all dynamic statement output and places this information into the Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
statement-path	statement-path	The absolute path to the statement nodes in the DART Event Type Definition.

Return Values

Boolean

Returns **#t** (true) when successful; otherwise **#f** (false).

Throws

None.

db-struct-fetch

Syntax

```
(db-struct-fetch connection-handle table-path)
```

Description

db-struct-fetch composes and executes an SQL FETCH statement according to the information and data carried under the table-path node of an Event Type Definition, and stores the return column values inside each of the column nodes.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	Event path	A table node of a DART Event Type Definition.

Return Values

Path
Returns the table path if the execution of the SQL FETCH statement is successful, or

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve error message.

Throws

None.

Examples

```
(define xlate
    (let ((input ($make-event-map in-delm in-struct))
          (output ($make-event-map out-delm out-struct)))
          (lambda ($make-event-string)
              ($event-parse input $make-event-string)
              ($event-clear output)
              (begin
                  (if (db-struct-select hdbc ~output%out.dbo.table2)
                      (do ((result "") ((boolean? result))
                          (set! result (db-struct-fetch hdbc
~output%out.dbo.table2))
                          (if (boolean? result))
                              (if (not result)
                                  (begin
                                      (display "db-struct-fetch
failed!\n")
                                     (display (db-get-error-str hdbc))
                                      (newline)
                                  )
                                  (begin
                                      ...
                                  )
                              )
                              (begin
```

```
                                        (display result)
                                        (newline)
                                    )
                                )
                            )
                        )
                        ...
                        (insert "" ~output%out "")
                    )
                )
            )
        )
```

Explanation

The example above shows a typical code segment of a Collaboration Rule that uses the Event Type Definition. In this example, the output defined by **out.ssc** is an Event Type Definition. After clearing the Output Event-string with the Output Event Type Definition, the Collaboration procedure uses **db-struct-select** to issue an SQLSELECT statement based on the information carried under Event- path [~output%out.dbo.table2].

It repeatedly uses **db-struct-fetch** to issue the SQL FETCH statement and stores the resulting column values inside each column node under the table path [~output%out.dbo.table2] until there are no more records to fetch.

db-struct-insert

Syntax

```
(db-struct-insert connection-handle table-path)
```

Description

db-struct-insert composes and executes an SQL INSERT statement according to the information and data carried under the **table-path** node of a DART Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	Event path	A table node of a DART Event Type Definition.

Return Values

Boolean

Returns **#t** (true) if the execution of the SQL INSERT statement is successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define xlate
    (let ((input ($make-event-map in-delm in-struct))
          (output ($make-event-map out-delm out-struct)))
         (lambda ($make-event-string)
             ($event-parse input $make-event-string)
             ($event-clear output)
             (begin
                 (if (db-struct-insert hdbc ~input%in.dbo.table2)
                     (begin
                         ...
                     )
                     (begin
                         (display (db-get-error-str hdbc))
                         (newline)
                     )
                 )
                 ...
                 (insert "" ~output%out "")
             )
         )
    )
)
```

Explanation

The example above shows a typical code segment of a Collaboration Rule that uses the Event Type Definition. In this example, the input defined by **in.ssc** is an Event Type

Definition. After parsing the Input Event-string with the Input Event Type Definition, the Collaboration procedure uses **db-struct-insert** to issue an SQL INSERT statement based on the information carried under the Event-path [~input%in.dbo.table2].

db-struct-select

Syntax

```
(db-struct-select connection-handle table-path where-clause)
```

Description

db-struct-select composes and executes an SQL SELECT statement according to the information and data carried under the table-path node of a DART Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	Event path	A table node of a DART Event Type Definition.
where-clause	string	The where clause used by the SQL SELECT statement.

Return Values

Boolean

Returns **#t** (true) if the execution of the SQL SELECT statement is successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Notes

1. Both **db-struct-select**, and **db-struct-fetch** use the same algorithm to generate the selection name for the **db-sql-select** and **db-sql-fetch** procedure call. If the table path is a table node under an owner (schema) node the selection name will be **owner.table**.
2. If the table path does not have an owner node above it, the selection name will be **table**. You must issue a **db-sql-fetch-cancel** call with either **owner.table** or **table** as the selection name, if you want to cancel the selection.

Note: *Important to use the exact table name previously used in structure to cancel, including case.*

Examples

```
(define xlate
    (let ((input ($make-event-map in-delm in-struct))
          (output ($make-event-map out-delm out-struct)))
         (lambda ($make-event-string)
             ($event-parse input $make-event-string)
             ($event-clear output)
             ($event-parse output (event->string output))
             (begin
```

```
                    (if (db-struct-select hdbc ~output%out.dbo.table2 "ID
= 5")
                         (begin
                            (db-struct-fetch hdbc ~output%out.dbo.table2)
                             ...
                               (db-sql-fetch-cancel hdbc "dbo.table2")
                         )
                         (begin
                             (display (db-get-error-str hdbc))
                           (newline)
                         )
                     )
                     ...
                     (insert "" ~output%out "")
                 )
             )
         )
     )
```

Explanation

The example above shows a typical code segment of a Collaboration Rules file that uses the Event Type Definition. In this example, the output defined by **out.ssc** is an Event Type Definition. After clearing the Output Event-string, the Collaboration Service uses **db-struct-select** to issue an SQL SELECT statement based on the information carried under the Event-path [~output%out.dbo.table2]. The selection was cancelled by **db-sql-fetch-cancel** with **dbo.table2** as the selection name.

db-struct-update

Syntax

```
(db-struct-update connection-handle table-path where-clause)
```

Description

db-struct-update composes and executes an SQL UPDATE statement according to the information and data carried under the table-path node of an Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	Event path	A table node of a DART Event Type Definition
where-clause	string	A where clause used by the SQL SELECT statement.

Return Values

Boolean

Returns **#t** (true) if the execution of the SQL UPDATE statement is successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define xlate
    (let ((input ($make-event-map in-delm in-struct))
          (output ($make-event-map out-delm out-struct)))
         (lambda ($make-event-string)
             ($event-parse input $make-event-string)
             ($event-clear output)
             (begin
                (if (db-struct-update hdbc ~input%in.dbo.table2 "ID =
5")
                      (begin
                          ...
                      )
                      (begin
                          (display (db-get-error-str hdbc))
                          (newline)
                      )
                 )
                 ...
                 (insert "" ~output%out "")
             )
         )
    )
)
```

Explanation

The example above shows a typical code segment of a Collaboration Rule that uses the Event Type Definition. In this example, the input defined by **in.ssc** is an Event Type Definition. After parsing the Input Event-string with the Input Event Type Definition, the Collaboration procedure uses **db-struct-update** to issue an SQL UPDATE statement based on the information carried under the Event-path [~input%in.dbo.table2].

5.8 Sample Monk Scripts

This section includes sample Monk scripts which demonstrate how to use the Oracle e*Way's Monk functions. These Monk scripts demonstrate the following activities:

- **"Initializing Monk Extensions" on page 193**
- **"Calling Stored Procedures" on page 194**
- **"Inserting Records with Dynamic SQL Statements" on page 196**
- **"Updating Records with Dynamic SQL Statements" on page 198**
- **"Selecting Records with Dynamic SQL Statements" on page 200**
- **"Deleting Records with Dynamic SQL Statements" on page 202**
- **"Inserting a Binary Image to a Database" on page 203**
- **"Retrieving an Image from a Database" on page 206**
- **"Common Supporting Routines" on page 208**

5.8.1 Initializing Monk Extensions

The sample script shows how to initialize the Monk extensions. This function is used by many of the other sample Monk scripts shown in this chapter.

To use this sample script in an actual implementation, modify the following values:

- **EGATE** – This designates the location of the e*Gate client.
- **dsn** – This is he name of the data source.
- **uid** – This is the user name.
- **pwd** – This is the login password.

```
;demo-init.monk

(define EGATE "/eGate/client")

; routine to load DART Monk extension
(define (load-library extension)
   (define filename (string-append EGATE "/bin/" extension))
   (if (file-exists? filename)
      (load-extension filename)
      (begin
         (display (string-append "File " filename " does not
exist.\n"))
         (abort filename)
      )
   )
)

(load-library "stc_monkext.dll")

;;
;; define STCDB variables, data source, user ID, and password
;;

(define STCDB "ORACLE8")

(load-library "stc_dbmonkext.dll")

(define dsn "database")
(define uid "Administrator")
(define pwd (encrypt-password uid "password"))
```

5.8.2 Calling Stored Procedures

This script gives an example of calling Stored Procedures. See **"Stored Procedure Functions" on page 149** for more details.

```
;demo-proc-execute.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; call stored procedure and display results
(define (execute-procedure hdbc hstmt)
   (let ((prm-count (db-proc-param-count hdbc hstmt)))
      (if (db-proc-execute hdbc hstmt)
         (begin
            (do ((col-count (db-proc-column-count hdbc hstmt) (db-
proc-column-count hdbc hstmt)))
                ((or (not (number? col-count)) (= col-count 0)))
               (display-proc-column-property hdbc hstmt col-count)
               (display-proc-column-value hdbc hstmt col-count)
            )
           (display-proc-parameter-output-value hdbc hstmt prm-count)
            (if (db-proc-return-exist hdbc hstmt)
               (begin
                  (display "return: value = ")
                  (display (db-proc-return-value hdbc hstmt))
                  (newline)
               )
            )
         )
         (display (db-get-error-str hdbc))
      )
   )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
   (display (db-get-error-str hdbc))
)

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "\ndatabase login succeed !\n")

      ; bind the stored procedure
      (define hstmt1 (bind-procedure hdbc "PERSONNEL.GET_EMPLOYEES"))

      ; call the stored procedure if the binding is successful
      (if (statement-handle? hstmt1)
         (begin
            (display "call PERSONNEL.GET_EMPLOYEES to get all sales
...\n\n")
            (if (and
                  (db-proc-param-assign hdbc hstmt1 0 "30")
                  (db-proc-param-assign hdbc hstmt1 1 "10")
                )
               (execute-procedure hdbc hstmt1)
               (display (db-get-error-str hdbc))
            )
         )
      )
```

```
            (if (not (db-logout hdbc))
               (display (db-get-error-str hdbc))
            )
         )
         (display (db-get-error-str hdbc))
      )
```

5.8.3 Inserting Records with Dynamic SQL Statements

```
;demo-stmt-insert.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
   (if (db-stmt-execute hdbc hstmt)
      (begin
         (display-stmt-row-count hdbc hstmt)
         #t
      )
      #f
   )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
   (display (db-get-error-str hdbc))
)

(define stmt1 "INSERT INTO SCOTT.BONUS SELECT ENAME, JOB, SAL, COMM
FROM SCOTT.EMP WHERE DEPTNO = ?")

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nInsert accounting department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "10")
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the insertions ...\n")
                     (if (not (db-commit hdbc))
                        (display (db-get-error-str hdbc))
                     )
                  )
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )

            (display "\nInsert sales department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "20")
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the insertions ...\n")
                     (if (not (db-commit hdbc))
                        (display (db-get-error-str hdbc))
                     )
                  )
                  (display (db-get-error-str hdbc))
```

```
                    )
                    (display (db-get-error-str hdbc))
                )
            )
        )

        (if (not (db-logout hdbc))
            (display (db-get-error-str hdbc))
        )
    )
    (display (db-get-error-str hdbc))
)
```

5.8.4 Updating Records with Dynamic SQL Statements

```
;demo-stmt-update.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
   (if (db-stmt-execute hdbc hstmt)
      (begin
         (display-stmt-row-count hdbc hstmt)
         #t
      )
      #f
   )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
   (display (db-get-error-str hdbc))
)

(define stmt1 "UPDATE SCOTT.BONUS SET COMM = ? WHERE JOB = ?")

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nUpdate commission of manager ...\n")
            (if
               (and
                  (db-stmt-param-assign hdbc hstmt1 0 "10")
                  (db-stmt-param-assign hdbc hstmt1 1 "MANAGER")
               )
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the updates ...\n")
                     (if (not (db-commit hdbc))
                        (display (db-get-error-str hdbc))
                     )
                  )
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )

            (display "\nUpdate commission of clerk ...\n")
            (if
               (and
                  (db-stmt-param-assign hdbc hstmt1 0 "20")
                  (db-stmt-param-assign hdbc hstmt1 1 "CLERK")
               )
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the updates ...\n")
```

```
                        (if (not (db-commit hdbc))
                           (display (db-get-error-str hdbc))
                        )
                     )
                     (display (db-get-error-str hdbc))
                  )
                  (display (db-get-error-str hdbc))
               )
            )
         )

         (if (not (db-logout hdbc))
            (display (db-get-error-str hdbc))
         )
      )
      (display (db-get-error-str hdbc))
   )
```

5.8.5 Selecting Records with Dynamic SQL Statements

```
;demo-stmt-select.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
   (if (db-stmt-execute hdbc hstmt)
      (begin
         (display-stmt-column-value hdbc hstmt)
         #t
      )
      #f
   )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
   (display (db-get-error-str hdbc))
)

(define stmt1 "SELECT EMPNO, ENAME, JOB FROM SCOTT.EMP WHERE JOB = ?")
(define stmt2 "SELECT ENAME, DNAME, JOB, HIREDATE FROM SCOTT.EMP,
SCOTT.DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO AND DEPT.DNAME = ?")

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statements
      (define hstmt1 (bind-statement hdbc stmt1))
      (define hstmt2 (bind-statement hdbc stmt2))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nList all salesman ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "SALESMAN")
               (if (not (execute-statement hdbc hstmt1))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
            (display "\nList all manager ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "MANAGER")
               (if (not (execute-statement hdbc hstmt1))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
         )
      )

      (if (statement-handle? hstmt2)
         (begin
            (display "\nList employee of accounting department
...\n\n")
            (if (db-stmt-param-assign hdbc hstmt2 0 "ACCOUNTING")
               (if (not (execute-statement hdbc hstmt2))
                  (display (db-get-error-str hdbc))
```

```
                     )
                     (display (db-get-error-str hdbc))
                  )
               )
               (display (db-get-error-str hdbc))
            )

            (if (not (db-logout hdbc))
               (display (db-get-error-str hdbc))
            )
         )
         (display (db-get-error-str hdbc))
      )
```

5.8.6 Deleting Records with Dynamic SQL Statements

```
;demo-stmt-delete.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
   (if (db-stmt-execute hdbc hstmt)
      (begin
         (display-stmt-row-count hdbc hstmt)
         #t
      )
      #f
   )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
   (display (db-get-error-str hdbc))
)

(define stmt1 "DELETE FROM SCOTT.BONUS WHERE ENAME IS NOT NULL")

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nDelete records from scott.bonus table ...\n")
            (if (execute-statement hdbc hstmt1)
               (begin
                  (display "\nCommit the deletions ...\n")
                  (if (not (db-commit hdbc))
                     (display (db-get-error-str hdbc))
                  )
               )
               (display (db-get-error-str hdbc))
            )
         )
      )

      (if (not (db-logout hdbc))
         (display (db-get-error-str hdbc))
      )
   )
   (display (db-get-error-str hdbc))
)
```

5.8.7 Inserting a Binary Image to a Database

This sample shows how to insert a Binary Image into a Database. It uses both Static and Dynamic SQL functions. See **"Static SQL Functions" on page 119** and **"Dynamic SQL Functions" on page 135** for more details.

```
;demo-image-insert.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

(define (query-exist hdbc hstmt id)
   (let ((rec-count 0) (result '#()))
      (if (db-stmt-param-assign hdbc hstmt 0 id)
         (if (db-stmt-execute hdbc hstmt)
            (begin
              (set! result (vector-ref (db-stmt-fetch hdbc hstmt) 0))
               (set! rec-count (string->number result))
               (set! result (db-stmt-fetch-cancel hdbc hstmt))
               (if (> rec-count 0)
                  (begin
                     (display "image already exist\n")
                     #t
                  )
                  #f
               )
            )
            (begin
               (display (db-get-error-str hdbc))
               #f
            )
         )
         (begin
            (display (db-get-error-str hdbc))
            #f
         )
      )
   )
)

(define (execute-statement hdbc hstmt)
   (let ((col-count (db-stmt-column-count hdbc hstmt)) (row-count 0))
      (if (db-stmt-execute hdbc hstmt)
         (begin
            (if (> col-count 0)
               (if (not (display-stmt-column-value hdbc hstmt col-
count))
                  (display (db-get-error-str hdbc))
               )
            )
            (set! row-count (db-stmt-row-count hdbc hstmt))
            (if (boolean? row-count)
               (display (db-get-error-str hdbc))
               (display (string-append "number of image insert = "
(number->string row-count) "\n"))
            )
            (newline)
            #t
         )
         #f
      )
   )
```

```
)

(define (bind-image-statement hdbc stmt)
   (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
      (display (string-append "\nDynamic statement : " stmt "\n"))
      (if (statement-handle? hstmt)
         (begin
;            (db-stmt-param-bind hdbc hstmt 0 "SQL_INTEGER" 4 0)
;            (db-stmt-param-bind hdbc hstmt 1 "SQL_VARCHAR" 20 0)
;            (db-stmt-param-bind hdbc hstmt 2 "SQL_VARCHAR" 10 0)
;            (db-stmt-param-bind hdbc hstmt 3 "SQL_INTEGER" 38 0)
;            (db-stmt-param-bind hdbc hstmt 4 "SQL_INTEGER" 38 0)
;            (db-stmt-param-bind hdbc hstmt 5 "SQL_INTEGER" 10 0)
            (db-stmt-param-bind hdbc hstmt 6 "SQL_LONGVARBINARY"
2000000 0)
            (define prm-count (db-stmt-param-count hdbc hstmt))
            (display-stmt-parameter-property hdbc hstmt prm-count)

            (define col-count (db-stmt-column-count hdbc hstmt))
            (display-stmt-column-property hdbc hstmt col-count)
          )
          (display (db-get-error-str hdbc))
      )
      hstmt
   )
)


(define image1-id "7100")
(define image1-name "Coast")
(define image1-type "JPEG")
(define image1-width "1280")
(define image1-height "1024")
(define image1-file (string-append image1-name ".jpg"))

(define image-port (open-input-file image1-file))
(define image1-data (read image-port 1000000))
(close-port image-port)
(define image1-size (number->string (string-length image1-data)))

(define image2-id "7200")
(define image2-name "Launch")
(define image2-type "JPEG")
(define image2-width "2000")
(define image2-height "1600")
(define image2-file (string-append image2-name ".jpg"))

(define image-port (open-input-file image2-file))
(define image2-data (read image-port 2000000))
(close-port image-port)
(define image2-size (number->string (string-length image2-data)))

(define hdbc (make-connection-handle))
(display (connection-handle? hdbc)) (newline)

(define stmt0 "select count(0) from SCOTT.IMAGE where PIX_ID = ?")
(define stmt1 "insert into SCOTT.IMAGE (PIX_ID, PIX_NAME, PIX_TYPE,
BYTE_SIZE, PIX_WIDTH, PIX_HEIGHT, PIX_DATA) values (?, ?, ?, ?, ?, ?,
?)")

(if (db-login hdbc dsn uid pwd)
  (begin
    (display "\ndatabase login succeed !\n")
    (display (db-dbms hdbc)) (newline)
```

```
      (display (db-std-timestamp-format hdbc)) (newline)
      (display (db-max-long-data-size hdbc 2000000)) (newline)

      ; bind the query and insert statement
      (define hquery  (bind-statement hdbc stmt0))
      (define hinsert (bind-image-statement hdbc stmt1))

      (if (and
         (statement-handle? hquery)
         (statement-handle? hinsert)
       )
       (begin
         (if (not (query-exist hdbc hquery image1-id))
           (begin
             (display (string-append "insert image " image1-file "\n"))
             (if (and
                 (db-stmt-param-assign hdbc hinsert 0 image1-id)
                 (db-stmt-param-assign hdbc hinsert 1 image1-name)
                 (db-stmt-param-assign hdbc hinsert 2 image1-type)
                 (db-stmt-param-assign hdbc hinsert 3 image1-size)
                 (db-stmt-param-assign hdbc hinsert 4 image1-width)
                 (db-stmt-param-assign hdbc hinsert 5 image1-height)
                 (db-stmt-param-assign hdbc hinsert 6 image1-data)
               )
               (if (execute-statement hdbc hinsert)
                 (db-commit hdbc)
                 (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
             )
           )
         )

         (if (not (query-exist hdbc hquery image2-id))
           (begin
             (display (string-append "insert image " image2-file "\n"))
             (if (and
                 (db-stmt-param-assign hdbc hinsert 0 image2-id)
                 (db-stmt-param-assign hdbc hinsert 1 image2-name)
                 (db-stmt-param-assign hdbc hinsert 2 image2-type)
                 (db-stmt-param-assign hdbc hinsert 3 image2-size)
                 (db-stmt-param-assign hdbc hinsert 4 image2-width)
                 (db-stmt-param-assign hdbc hinsert 5 image2-height)
                 (db-stmt-param-assign hdbc hinsert 6 image2-data)
               )
               (if (execute-statement hdbc hinsert)
                 (db-commit hdbc)
                 (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
             )
           )
         )
       )
      )

      (if (not (db-logout hdbc))
        (display (db-get-error-str hdbc))
      )
    )
    (display (db-get-error-str hdbc))
  )
```

5.8.8 Retrieving an Image from a Database

This sample shows how to Retrieve an image from a Database. It uses both Static and Dynamic SQL functions. See **"Static SQL Functions" on page 119** and **"Dynamic SQL Functions" on page 135** for more details.

```
;demo-image-select.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

(define (get-image hdbc hstmt)
   (do (
         (result (db-stmt-fetch hdbc hstmt) (db-stmt-fetch hdbc
hstmt))
         (first_name "")
         (file_type "")
         (file_name "")
         (width "")
         (height "")
         (output_port '())
       )
       ((boolean? result) result)
      (set! first_name (vector-ref result 0))
      (set! file_type (strip-trailing-whitespace (vector-ref result
1)))
      (set! width (strip-trailing-whitespace (vector-ref result 2)))
      (set! height (strip-trailing-whitespace (vector-ref result 3)))
      (cond
         ((string=? file_type "JPEG") (set! file_name (string-append
first_name ".jpg")))
         ((string=? file_type "GIF") (set! file_name (string-append
first_name ".gif")))
        ((string=? file_type "BITMAP") (set! file_name (string-append
first_name ".bmp")))
         ((string=? file_type "TIFF") (set! file_name (string-append
first_name ".tif")))
         (else (set! file_name (string-append first_name ".raw")))
      )
      (if (file-exists? file_name)
         (file-delete file_name)
      )
      (display (string-append "picture name = " file_name "\n"))
      (display (string-append "picture size = " width " x " height
"\n\n"))
      (set! output_port (open-output-file file_name))
      (display (vector-ref result 4) output_port)
      (close-port output_port)
   )
)

(define (execute-statement hdbc hstmt)
   (let ((col-count (db-stmt-column-count hdbc hstmt)) (row-count 0))
      (if (db-stmt-execute hdbc hstmt)
         (begin
            (if (> col-count 0)
               (if (not (get-image hdbc hstmt))
                  (display (db-get-error-str hdbc))
               )
            )
            (set! row-count (db-stmt-row-count hdbc hstmt))
            (if (boolean? row-count)
```

```
                (display (db-get-error-str hdbc))
               (display (string-append "number of image retrieved = "
(number->string row-count) "\n"))
            )
            (newline)
            #t
         )
         #f
      )
   )
)

(define hdbc (make-connection-handle))
(display (connection-handle? hdbc)) (newline)

(define stmt "select PIX_NAME, PIX_TYPE, PIX_WIDTH, PIX_HEIGHT,
PIX_DATA from SCOTT.IMAGE where PIX_ID = ?")

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "\ndatabase login succeed !\n")
      (display (db-dbms hdbc)) (newline)
      (display (db-std-timestamp-format hdbc)) (newline)
      (display (db-max-long-data-size hdbc 2000000)) (newline)

      ; bind the select statement
      (define hselect (bind-binary-statement hdbc stmt))

      ; execute the dynamic statement
      (display "select IMAGE table\n")
      (if (statement-handle? hselect)
         (begin
            (if (db-stmt-param-assign hdbc hselect 0 "7100")
               (if (not (execute-statement hdbc hselect))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
            (if (db-stmt-param-assign hdbc hselect 0 "7200")
               (if (not (execute-statement hdbc hselect))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
         )
      )

      (if (not (db-logout hdbc))
         (display (db-get-error-str hdbc))
      )
   )
   (display (db-get-error-str hdbc))
)
```

5.8.9 Common Supporting Routines

This sample script displays and defines values and parameters for stored procedures. The routines contained in this script are used by many of the Monk samples in this chapter. For more details about functions used in this script, see **"Stored Procedure Functions" on page 149**

```
;demo-common.monk

;;
;; stored procedure auxiliary functions
;;

; display parameter properties of the stored procedure
(define (display-proc-parameter-property hdbc hstmt prm-count)
   (display "parameter count = ") (display prm-count) (newline)
   (do ((i 0 (+ i 1))) ((= i prm-count))
      (display "parameter ")
      (display (db-proc-param-name hdbc hstmt i))
      (display ": type = ")
      (display (db-proc-param-type hdbc hstmt i))
      (display ", io = ")
      (display (db-proc-param-io hdbc hstmt i))
      (newline)
   )
)

; display value of output parameters from stored procedure
(define (display-proc-parameter-output-value hdbc hstmt prm-count)
   (do ((i 0 (+ i 1))) ((= i prm-count))
      (if (not (equal? (db-proc-param-io hdbc hstmt i) "IN"))
         (begin
            (display "output parameter ")
            (display (db-proc-param-name hdbc hstmt i))
            (display " = ")
            (display (db-proc-param-value hdbc hstmt i))
            (newline)
         )
      )
   )
)

; display column properties of the return result set
(define (display-proc-column-property hdbc hstmt col-count)
   (display "column count = ") (display col-count) (newline)
   (do ((i 0 (+ i 1))) ((= i col-count))
      (display "column ")
      (display (db-proc-column-name hdbc hstmt i))
      (display ": type = ")
      (display (db-proc-column-type hdbc hstmt i))
      (newline)
   )
   (newline)
)

; display column value of the return result set of the stored
procedure
(define (display-proc-column-value hdbc hstmt col-count)
   (define (fetch-next)
      (let ((result (db-proc-fetch hdbc hstmt)))
         (if (boolean? result)
            result
            (begin (display result) (newline) (fetch-next))
```

```
         )
      )
   )
   (fetch-next)
   (newline)
)

; bind stored procedure and display parameter properties
(define (bind-procedure hdbc proc)
   (let ((hstmt (db-proc-bind hdbc proc)))
      (if (statement-handle? hstmt)
         (begin
            (display (string-append "bind stored procedure : " proc
"\n"))
            (define prm-count (db-proc-param-count hdbc hstmt))
            (display-proc-parameter-property hdbc hstmt prm-count)
            (newline)
            (if (db-proc-return-exist hdbc hstmt)
               (begin
                  (display "return: type = ")
                  (display (db-proc-return-type  hdbc hstmt))
                  (newline)
               )
            )
            (newline)
         )
          (display (db-get-error-str hdbc))
      )
      hstmt
   )
)




;;
;; dynamic statement auxiliary functions
;;

; display parameter properties of the SQL statement
(define (display-stmt-parameter-property hdbc hstmt prm-count)
   (display "parameter count = ") (display prm-count) (newline)
   (do ((i 0 (+ i 1))) ((= i prm-count))
      (display "parameter #")
      (display i)
      (display ": type = ")
      (display (db-stmt-param-type hdbc hstmt i))
      (newline)
   )
   (newline)
)

; display column properties of the SQL statement
(define (display-stmt-column-property hdbc hstmt col-count)
   (display "column count = ") (display col-count) (newline)
   (do ((i 0 (+ i 1))) ((= i col-count))
      (display "column ")
      (display (db-stmt-column-name hdbc hstmt i))
      (display ": type = ")
      (display (db-stmt-column-type hdbc hstmt i))
      (newline)
   )
   (newline)
)
```

```
; display column value of the return result set of the SQL statement
(define (display-stmt-column-value hdbc hstmt)
   (define (fetch-next)
      (let ((result (db-stmt-fetch hdbc hstmt)))
         (if (boolean? result)
            result
            (begin (display result) (newline) (fetch-next))
         )
      )
   )
   (fetch-next)
   (newline)
)

; display row count affected by the execution of the SQL statement
(define (display-stmt-row-count hdbc hstmt)
   (let ((row-count (db-stmt-row-count hdbc hstmt)))
      (cond
         ((= row-count 0) (display "\n(no row affected)\n"))
         ((= row-count 1) (display "\n(1 row affected)\n"))
         (else (display (string-append "\n(" (number->string row-
count) " rows affected)\n")))
      )
   )
)

; bind dynamic statement and display paramters and column properties
(define (bind-statement hdbc stmt)
   (let ((hstmt (db-stmt-bind hdbc stmt)))
      (display (string-append "\nDynamic statement : " stmt "\n"))
      (if (statement-handle? hstmt)
         (begin
            (define prm-count (db-stmt-param-count hdbc hstmt))
            (display-stmt-parameter-property hdbc hstmt prm-count)

            (define col-count (db-stmt-column-count hdbc hstmt))
            (display-stmt-column-property hdbc hstmt col-count)
          )
          (display (db-get-error-str hdbc))
      )
      hstmt
   )
)

; bind dynamic statement to input/output raw binary data
(define (bind-binary-statement hdbc stmt)
   (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
      (display (string-append "\nDynamic statement : " stmt "\n"))
      (if (statement-handle? hstmt)
         (begin
            (define prm-count (db-stmt-param-count hdbc hstmt))
            (display-stmt-parameter-property hdbc hstmt prm-count)

            (define col-count (db-stmt-column-count hdbc hstmt))
            (display-stmt-column-property hdbc hstmt col-count)
          )
          (display (db-get-error-str hdbc))
      )
      hstmt
   )
)
```

Index

A

additional path **35**
auxiliary library directories **35**

B

basic e*Way processes **28**
basic functions **78**
  event-send-to-egate **79**
  get-logical-name **80**
  send-external-down **81**
  send-external-up **82**
  shutdown-request **83**
  start-schedule **84**
  stop-schedule **85**
build an event type **44**
build tool **44**

C

calling stored procedures, sample **194**
common supporting routines, sample **208**
communication setup **24**
components **9**
configuration **22**
configuration parameters **22**
connection-handle? **105**
converter, DART **44**
creating database user account **21**

D

DART **21**, **22**
  converter **44**
  library **45**
data exchange functions
  event-driven **33**
  schedule-driven **31**
database management system **9**
database name **42**
database setup **41**
  database name **42**
  database type **41**
  encrypted password **42**
  user name **42**
database type **41**
db-alive **106**
db-commit **108**
db-get-error-str **109**
db-login **111**
db-logout **113**
db-max-long-data-size **114**
DBMS **9**
db-proc-bind **151**
db-proc-bind-binary **152**
db-proc-column-count **153**
db-proc-column-name **155**
db-proc-column-type **157**
db-proc-execute **159**
db-proc-fetch function **161**
db-proc-fetch-cancel **163**
db-proc-max-records **164**
db-proc-param-assign **165**
db-proc-param-count **167**
db-proc-param-io **169**
db-proc-param-name **170**
db-proc-param-type **171**
db-proc-param-value **172**
db-proc-return-exist **174**
db-proc-return-type **176**
db-proc-return-value **178**
db-rollback **115**
db-sql-column-names **123**
db-sql-column-types **125**
db-sql-column-values **127**
db-sql-execute **129**
db-sql-fetch **130**
db-sql-fetch-cancel **131**
db-sql-format **132**
db-sql-select **134**
db-stdver-conn-estab **87**
db-stdver-conn-shutdown **89**
db-stdver-conn-ver **90**
db-stdver-data-exchg **92**
db-stdver-data-exchg-stub **93**
db-stdver-init **94**
db-stdver-neg-ack **96**
db-stdver-pos-ack **97**
db-stdver-proc-outgoing **98**
db-stdver-proc-outgoing-stub **100**
db-stdver-shutdown **102**
db-stdver-startup **103**
db-stmt-bind **136**
db-stmt-bind-binary **137**
db-stmt-column-count **138**
db-stmt-column-name **139**
db-stmt-column-type **140**
db-stmt-execute **141**

db-stmt-fetch **142**
db-stmt-fetch-cancel **143**
db-stmt-param-assign **144**
db-stmt-param-bind **145**
db-stmt-param-count **146**
db-stmt-param-type **147**
db-stmt-row-count **148**
db-struct-bulk-insert **181**
db-struct-call **182**
db-struct-execute **183**
db-struct-fetch **184**
db-struct-insert **186**
db-struct-select **188**
db-struct-update **190**
deleting records, sample **202**
down timeout **26**
dynamic SQL functions **119**, **135**
  db-stmt-bind **136**
  db-stmt-bind-binary **137**
  db-stmt-column-count **138**
  db-stmt-column-name **139**
  db-stmt-column-type **140**
  db-stmt-execute **141**
  db-stmt-fetch **142**
  db-stmt-fetch-cancel **143**
  db-stmt-param-assign **144**
  db-stmt-param-bind **145**
  db-stmt-param-count **146**
  db-stmt-param-type **147**
  db-stmt-row-count **148**

E

e*Way configuration parameters **22**
encrypted password **42**
ETD editor’s build tool **44**
event-driven data exchange functions **33**
event-send-to-egate **79**
exchange data interval **25**
exchange data with external function **37**
external connection establishment function **38**
external connection shutdown function **39**
external connection verification function **39**
external system requirements **10**

F

forward external errors **24**
functions
  connection-handle **105**
  db-alive **106**
  db-commit **108**
  db-get-error-str **109**
  db-login **111**
  db-logout **113**
  db-max-long-data-size **114**
  db-proc-bind **151**
  db-proc-bind-binary **152**
  db-proc-column-count **153**
  db-proc-column-name **155**
  db-proc-column-type **157**
  db-proc-execute **159**
  db-proc-fetch **161**
  db-proc-fetch-cancel **163**
  db-proc-max-records **164**
  db-proc-param-assign **165**
  db-proc-param-count **167**
  db-proc-param-io **169**
  db-proc-param-name **170**
  db-proc-param-type **171**
  db-proc-param-value **172**
  db-proc-return-exist **174**
  db-proc-return-type **176**
  db-proc-return-value **178**
  db-rollback **115**
  db-sql-column-names **123**
  db-sql-column-types **125**
  db-sql-column-values **127**
  db-sql-execute **129**
  db-sql-fetch **130**
  db-sql-fetch-cancel **131**
  db-sql-format **132**
  db-sql-select **134**
  db-std-timestamp-format **116**
  db-stdver-conn-estab **87**
  db-stdver-conn-shutdown **89**
  db-stdver-conn-ver **90**
  db-stdver-data-exchg **92**
  db-stdver-data-exchg-stub **93**
  db-stdver-init **94**
  db-stdver-neg-ack **96**
  db-stdver-pos-ack **97**
  db-stdver-proc-outgoing **98**
  db-stdver-proc-outgoing-stub **100**
  db-stdver-shutdown **102**
  db-stdver-startup **103**
  db-stmt-bind **136**
  db-stmt-bind-binary **137**
  db-stmt-column-count **138**
  db-stmt-column-name **139**
  db-stmt-column-type **140**
  db-stmt-execute **141**
  db-stmt-fetch **142**
  db-stmt-fetch-cancel **143**
  db-stmt-param-assign **144**
  db-stmt-param-bind **145**
  db-stmt-param-count **146**
  db-stmt-param-type **147**

db-stmt-row-count **148**
db-struct-bulk-insert **181**
db-struct-call **182**
db-struct-execute **183**
db-struct-fetch **184**
db-struct-insert **186**
db-struct-select **188**
db-struct-update **190**
event-send-to-egate **79**
get-logical-name **80**
make-connection-handle **117**
send-external-down **81**
send-external-up **82**
shutdown-request **83**
start-schedule **84**
statement-handle? **118**
stop-schedule **85**

G

general connection functions **104**
connection-handle? **105**
db-alive **106**
db-commit **108**
db-get-error-str **109**
db-login **111**
db-logout **113**
db-max-long-data-size **114**
db-rollback **115**
db-std-timestamp-format **116**
make-connection-handle **117**
statement-handle? **118**
general settings **23**
get-logical-name function **80**

I

IDN **9**
implementation **44**
initializing Monk extensions, sample **193**
inserting records, sample **196**
Installation
UNIX **14**
installation **11**
client **12**
decisions **11**
files **14**, **16**
network components **12**
overview **11**
pre-installation **14**
procedure **13**, **15**
troubleshooting **19**
Windows **12**
integrated delivery network **9**
intended reader **9**
introduction **8**

J

journal file name **23**

L

LD_LIBRARY_PATH **42**
library converter **44**
library, DART **45**
listener **19**
listener.ora **17**

M

make-connection-handle **117**
max failed messages **24**
max resends per message **23**
message event functions **180**
db-struct-bulk-insert **181**
db-struct-call **182**
db-struct-execute **183**
db-struct-fetch **184**
db-struct-insert **186**
db-struct-select **188**
db-struct-update **190**
monk
notes **35**
monk environment initialization file **35**

N

negative acknowledgment function **40**
notes on monk **35**

O

OCI **12**, **16**
Oracle Call Interface **12**, **16**
oracle e*Way functions **78**
oracle SQL type support **122**
ORACLE_HOME **42**
ORACLE_SID **42**

P

parameters
additional path **35**
auxiliary library directories **35**
communication setup **24**
configuration **22**

database name **42**
database setup **41**
database type **41**
down timeout **26**
encrypted password **42**
exchange data interval **25**
exchange data with external function **37**
external connection establishment function **38**
external connection shutdown function **39**
external connection verification function **39**
forward external errors **24**
general settings **23**
journal file name **23**
max failed messages **24**
max resends per message **23**
monk environment initialization file **35**
negative acknowledgment function **40**
positive acknowledgment function **40**
process outgoing message function **37**
resend timeout **26**
shutdown command notification function **41**
start exchange data schedule **24**
startup function **36**
stop exchange data schedule **25**
up timeout **26**
user name **42**
zero wait between successful exchanges **26**
positive acknowledgment function **40**
process outgoing message function **37**
publishing to an Oracle database, sample **53**

R

requirements
external configuration **42**
external system **10**
system **10**
resend timeout **26**

S

sample
calling stored procedures **194**
common routines **208**
common supporting routines **208**
deleting records with dynamic SQL statements **202**
dynamic SQL statements **196**, **198**, **200**, **202**
initializing Monk extensions **193**
inserting binary images **203**
inserting records with dynamic SQL statements **196**
Monk scripts **192**
publishing to an Oracle database **53**
retrieving images **206**
selecting records with dynamic SQL statements **200**
stored procedures **194**
updating records with dynamic SQL statements **198**
sample Monk scripts **192**
schedule-driven data exchange functions **31**
search path, shared library **20**
selecting records, sample **200**
send-external-down function **81**
send-external-up function **82**
shared library **20**
shutdown command notification function **41**
Shutdown Functions **33**
shutdown-request **83**
specify file names **34**
specify function names **34**
SQL **9**
SQL*Net **12**, **16**, **19**
SQL92 standard format **116**
sqlnet.ora **17**
standard e*Way functions **86**
db-stdver-conn-estab **87**
db-stdver-conn-shutdown **89**
db-stdver-conn-ver **90**
db-stdver-data-exchg **92**
db-stdver-data-exchg-stub **93**
db-stdver-init **94**
db-stdver-neg-ack **96**
db-stdver-pos-ack **97**
db-stdver-proc-outgoing **98**
db-stdver-proc-outgoing-stub **100**
db-stdver-shutdown **102**
db-stdver-startup **103**
start exchange data schedule **24**
starting a listener **19**
start-schedule function **84**
startup function **36**
statement-handle? **118**
static SQL functions **119**
db-sql-column-names **123**
db-sql-column-types **125**
db-sql-column-values **127**
db-sql-execute **129**
db-sql-fetch **130**
db-sql-fetch-cancel **131**
db-sql-format **132**
db-sql-select **134**
static vs. dynamic SQL functions **119**
stcewgenericmonk.exe **9**, **13**, **14**, **15**, **16**
stop exchange data schedule **25**
stop-schedule function **85**
stored procedure functions **149**

db-proc-bind **151**
db-proc-bind-binary **152**
db-proc-column-count **153**
db-proc-column-name **155**
db-proc-column-type **157**
db-proc-execute **159**
db-proc-fetch **161**
db-proc-fetch-cancel **163**
db-proc-max-records **164**
db-proc-param-assign **165**
db-proc-param-count **167**
db-proc-param-io **169**
db-proc-param-name **170**
db-proc-param-type **171**
db-proc-param-value **172**
db-proc-return-exist **174**
db-proc-return-type **176**
db-proc-return-value **178**
stored procedures, sample **194**
supported variable SQL data types **122**
system requirements **10**

T

TCP/IP **12**, **16**
testing
SQL*Net **19**
tnsnames.ora **17**

U

up timeout **26**
updating records, sample **198**
user account, creating **21**
user name **42**
using SQL **9**

Z

zero wait between successful exchanges **26**